

# ***TMS320C54x DSP Reference Set***

## ***Volume 2: Mnemonic Instruction Set***

Best Available Copy

Literature Number: SPRU172  
Manufacturing Part Number: D425007-9761 revision \*  
October 1996



Best Available Copy

### **IMPORTANT NOTICE**

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

**TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.**

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

## Preface

# Read This First

### ***About This Manual***

The TMS320C54x is a fixed-point digital signal processor (DSP) in the TMS320 family, and it can use either of two forms of the instruction set: a mnemonic form or an algebraic form. This book is a reference for the mnemonic form of the instruction set. It contains information about the instructions used for all types of operations (arithmetic, logical, load and store, conditional, and program control), the nomenclature used in describing the instruction operation, and supplemental information you may need, such as interrupt priorities and locations.

This book uses a shortened form of the device name, '54x, to refer to all members of the device family and as an aid in readability.

### ***How to Use This Manual***

The following table summarizes the '54x information contained in this book:

<b>If you are looking for information about:</b>	<b>Turn to:</b>
Arithmetic operations	Chapter 2, <i>Instruction Set Summary</i>
Conditions for conditional instructions	Appendix A, <i>Condition Codes</i>
Example description of instruction	Chapter 1, <i>Symbols and Abbreviations</i>
Individual instruction descriptions	Chapter 4, <i>Assembly Language Instructions</i>
Instruction set abbreviations	Chapter 1, <i>Symbols and Abbreviations</i>
Instruction set classes	Chapter 3, <i>Instruction Classes and Cycles</i>

If you are looking for information about:	Turn to:
Instruction set symbols	Chapter 1, <i>Symbols and Abbreviations</i>
Interrupt locations and priorities	Appendix B, <i>Interrupt Locations and Priority Tables</i>
Interrupt register layout	Appendix C, <i>Interrupt and Status Registers</i>
Load and store operations	Chapter 2, <i>Instruction Set Summary</i>
Logical operations	Chapter 2, <i>Instruction Set Summary</i>
Program control operations	Chapter 2, <i>Instruction Set Summary</i>
Status register layout	Appendix C, <i>Interrupt and Status Registers</i>
Summary of instructions	Chapter 2, <i>Instruction Set Summary</i>

## Notational Conventions

This book uses the following conventions.

- Program listings and program examples are shown in a special typeface.

Here is a segment of a program listing:

```
LMS *AR3+, *AR4+
```

- In syntax descriptions, the instruction is in a **bold typeface** and parameters are in an *italic typeface*. Portions of a syntax in **bold** must be entered as shown; portions of a syntax in *italics* describe the type of information that you specify. Here is an example of an instruction syntax:

**LMS** *Xmem, Ymem*

**LMS** is the instruction, and it has two parameters, *Xmem* and *Ymem*. When you use **LMS**, the parameters should be actual dual data-memory operand values. A comma and a space (optional) must separate the two values.

- The term OR is used in the assembly language instructions to denote a Boolean operation. The term or is used to indicate selection. Here is an example of an instruction with OR and or:

```
lk OR (src) → src or [dst]
```

This instruction ORs the value of lk with the contents of src. Then, it stores the result in src or dst, depending on the syntax of the instruction.



- Square brackets, [ and ], identify an optional parameter. If you use an optional parameter, specify the information within the brackets; do not type the brackets themselves.

## **Related Documentation From Texas Instruments**

The following books describe the '54x and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number.

The **TMS320C54x DSP Reference Set** (literature number SPRU210) is composed of four volumes of information, each with its own literature number for individual ordering.

**TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals** (literature number SPRU131) describes the TMS320C54x 16-bit, fixed-point, general-purpose digital signal processors. Covered are its architecture, internal register structure, data and program addressing, the instruction pipeline, DMA, and on-chip peripherals. Also includes development support information, parts lists, and design considerations for using the XDS510 emulator.

**TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set** (literature number SPRU172) describes the TMS320C54x digital signal processor mnemonic instructions individually. Also includes a summary of instruction set classes and cycles.

**TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set** (literature number SPRU179) describes the TMS320C54x digital signal processor algebraic instructions individually. Also includes a summary of instruction set classes and cycles.

**TMS320C54x DSP Reference Set, Volume 4: Applications Guide** (literature number SPRU173) describes software and hardware applications for the TMS320C54x digital signal processor. Also includes development support information, parts lists, and design considerations for using the XDS510 emulator.

**TMS320C54x DSKplus User's Guide** (literature number SPRU191) describes the TMS320C54x digital signal processor starter kit (DSK), which allows you to execute custom 'C54x code in real time and debug it line by line. Covered are installation procedures, a description of the debugger and the assembler, customized applications, and initialization routines.

**TMS320C54x Assembly Language Tools User's Guide** (literature number SPRU102) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C54x generation of devices.

**TMS320C5xx C Source Debugger User's Guide** (literature number SPRU099) tells you how to invoke the 'C54x emulator, EVM, and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

**TMS320C54x Code Generation Tools Getting Started Guide** (literature number SPRU147) describes how to install the TMS320C54x assembly language tools and the C compiler for the 'C54x devices. The installation for MS-DOS™, OS/2™, SunOS™, Solaris™, and HP-UX™ 9.0x systems is covered.

**TMS320C54x Evaluation Module Technical Reference** (literature number SPRU135) describes the 'C54x EVM, its features, design details and external interfaces.

**TMS320C54x Optimizing C Compiler User's Guide** (literature number SPRU103) describes the 'C54x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C54x generation of devices.

**TMS320C54x Simulator Getting Started** (literature number SPRU137) describes how to install the TMS320C54x simulator and the C source debugger for the 'C54x. The installation for MS-DOS™, PC-DOS™, SunOS™, Solaris™, and HP-UX™ systems is covered.

**TMS320 Third-Party Support Reference Guide** (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of '320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

**TMS320 DSP Development Support Reference Guide** (literature number SPRU011) describes the TMS320 family of digital signal processors and the tools that support these devices. Included are code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). Also covered are available documentation, seminars, the university program, and factory repair and exchange.

***Trademarks***

HP-UX is a trademark of Hewlett-Packard Company.

MS-DOS is a registered trademark of Microsoft Corporation.

OS/2 and PC-DOS are trademarks of International Business Machines Corporation.

Solaris and SunOS are trademarks of Sun Microsystems, Inc.

TI is a trademark of Texas Instruments Incorporated.

**If You Need Assistance. . .**

<input type="checkbox"/> <b>World-Wide Web Sites</b>		
TI Online	<a href="http://www.ti.com">http://www.ti.com</a>	
Semiconductor Product Information Center (PIC)	<a href="http://www.ti.com/sc/docs/pic/home.htm">http://www.ti.com/sc/docs/pic/home.htm</a>	
DSP Solutions	<a href="http://www.ti.com/dsps">http://www.ti.com/dsps</a>	
320 Hotline On-line™	<a href="http://www.ti.com/sc/docs/dsps/support.html">http://www.ti.com/sc/docs/dsps/support.html</a>	
<input type="checkbox"/> <b>North America, South America, Central America</b>		
Product Information Center (PIC)	(972) 644-5580	
TI Literature Response Center U.S.A.	(800) 477-8924	
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285	
U.S. Technical Training Organization	(972) 644-5580	
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324    Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323	
DSP Internet BBS via anonymous ftp to <a href="ftp://ftp.ti.com/mirrors/tms320bbs">ftp://ftp.ti.com/mirrors/tms320bbs</a>		
<input type="checkbox"/> <b>Europe, Middle East, Africa</b>		
European Product Information Center (EPIC) Hotlines:		
Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32    Email: epic@ti.com
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10
<input type="checkbox"/> <b>Asia-Pacific</b>		
Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	
<input type="checkbox"/> <b>Japan</b>		
Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type "Go TIASP"	
<input type="checkbox"/> <b>Documentation</b>		
When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.		
Mail: Texas Instruments Incorporated	Email: <a href="mailto:comments@books.sc.ti.com">comments@books.sc.ti.com</a>	
Technical Documentation Services, MS 702		
P.O. Box 1443		
Houston, Texas 77251-1443		

**Note:** When calling a Literature Response Center to order documentation, please specify the literature number of the book.

# Contents

<b>1</b>	<b>Symbols and Abbreviations</b>	<b>1-1</b>
	<i>Lists and defines the symbols and abbreviations used in the instruction set summary and in the individual instruction descriptions. Also provides an example description of an instruction.</i>	
1.1	Instruction Set Symbols and Abbreviations	1-2
1.2	Example Description of Instruction	1-9
<b>2</b>	<b>Instruction Set Summary</b>	<b>2-1</b>
	<i>Provides a summary of the instruction set divided into four basic types of operation. Also includes information on repeating a single instruction and a list of nonrepeatable instructions.</i>	
2.1	Arithmetic Operations	2-2
2.2	Logical Operations	2-8
2.3	Program-Control Operations	2-10
2.4	Load and Store Operations	2-14
2.5	Repeating a Single Instruction	2-19
<b>3</b>	<b>Instruction Classes and Cycles</b>	<b>3-1</b>
	<i>Describes the classes and lists the cycles of the instruction set.</i>	
<b>4</b>	<b>Assembly Language Instructions</b>	<b>4-1</b>
	<i>Describes the '54x assembly language instructions individually.</i>	
<b>A</b>	<b>Condition Codes</b>	<b>A-1</b>
	<i>Lists the conditions used in conditional instructions and the combination of conditions that can be tested.</i>	
<b>B</b>	<b>Interrupt Locations and Priority Tables</b>	<b>B-1</b>
	<i>Lists the '54x interrupt locations and priorities for each individual device type.</i>	
<b>C</b>	<b>Interrupt and Status Registers</b>	<b>C-1</b>
	<i>Shows the bit fields of the '54x interrupt and status registers.</i>	
<b>D</b>	<b>Glossary</b>	<b>D-1</b>
	<i>Defines terms and abbreviations used throughout this book.</i>	

# Figures

C-1	Interrupt Flag Register (IFR) .....	C-3
C-2	Interrupt Mask Register (IMR) .....	C-4
C-3	Processor Mode Status Register (PMST) .....	C-5
C-4	Status Register 0 (ST0) .....	C-5
C-5	Status Register 1 (ST1) .....	C-5

# Tables

1-1	Instruction Set Symbols and Abbreviations .....	1-2
1-2	Opcode Symbols and Abbreviations .....	1-5
1-3	Instruction Set Notations .....	1-7
1-4	Operators Used in Instruction Set .....	1-8
2-1	Add Instructions .....	2-2
2-2	Subtract Instructions .....	2-3
2-3	Multiply Instructions .....	2-4
2-4	Multiply-Accumulate and Multiply-Subtract Instructions .....	2-4
2-5	Double (32-Bit Operand) Instructions .....	2-6
2-6	Application-Specific Instructions .....	2-7
2-7	AND Instructions .....	2-8
2-8	OR Instructions .....	2-8
2-9	XOR Instructions .....	2-9
2-10	Shift Instructions .....	2-9
2-11	Test Instructions .....	2-9
2-12	Branch Instructions .....	2-10
2-13	Call Instructions .....	2-11
2-14	Interrupt Instructions .....	2-11
2-15	Return Instructions .....	2-12
2-16	Repeat Instructions .....	2-12
2-17	Stack-Manipulating Instructions .....	2-13
2-18	Miscellaneous Program-Control Instructions .....	2-13
2-19	Load Instructions .....	2-14
2-20	Store Instructions .....	2-15
2-21	Conditional Store Instructions .....	2-16
2-22	Parallel Load and Store Instructions .....	2-16
2-23	Parallel Load and Multiply Instructions .....	2-16
2-24	Parallel Store and Add/Subtract Instructions .....	2-17
2-25	Parallel Store and Multiply Instructions .....	2-17
2-26	Miscellaneous Load-Type and Store-Type Instructions .....	2-18
2-27	Multicycle Instructions That Become Single-Cycle Instructions When Repeated .....	2-19
2-28	Nonrepeatable Instructions .....	2-20
A-1	Conditions for Conditional Instructions .....	A-2
A-2	Groupings of Conditions .....	A-3
B-1	'541 Interrupt Locations and Priorities .....	B-2
B-2	'542 Interrupt Locations and Priorities .....	B-3

## Tables

---

B-3	'543 Interrupt Locations and Priorities .....	B-4
B-4	'545 Interrupt Locations and Priorities .....	B-5
B-5	'546 Interrupt Locations and Priorities .....	B-6
B-6	'548 Interrupt Locations and Priorities .....	B-7
C-1	Register Field Terms and Definitions .....	C-1



# Symbols and Abbreviations

---

This chapter lists and defines the symbols and abbreviations used in the instruction set summary and in the individual instruction descriptions. It also provides an example description of an instruction.

Topic	Page
1.1 Instruction Set Symbols and Abbreviations .....	1-2
1.2 Example Description of Instruction .....	1-9

## 1.1 Instruction Set Symbols and Abbreviations

Table 1–1 through Table 1–4 list the symbols and abbreviations used in the instruction set summary (Chapter 2) and in the individual instruction descriptions (Chapter 4).

*Table 1–1. Instruction Set Symbols and Abbreviations*

Symbol	Meaning
A	Accumulator A
ALU	Arithmetic logic unit
AR	Auxiliary register, general usage
ARx	Designates a specific auxiliary register ( $0 \leq x \leq 7$ )
ARP	Auxiliary register pointer field in ST0; this 3-bit field points to the current auxiliary register (AR).
ASM	5-bit accumulator shift mode field in ST1 ( $-16 \leq ASM \leq 15$ )
B	Accumulator B
BRAF	Block-repeat active flag in ST1
BRC	Block-repeat counter
BITC	4-bit value that determines which bit of a designated data memory value is tested by the test bit instruction ( $0 \leq BITC \leq 15$ )
C16	Dual 16-bit/double-precision arithmetic mode bit in ST1
C	Carry bit in ST0
CC	2-bit condition code ( $0 \leq CC \leq 3$ )
CMPT	Compatibility mode bit in ST1
CPL	Compiler mode bit in ST1
cond	An operand representing a condition used by instructions that execute conditionally
[d], [D]	Delay option
DAB	D address bus
DAR	DAB address register
dmad	16-bit immediate data-memory address ( $0 \leq dmad \leq 65\,535$ )
Dmem	Data-memory operand
DP	9-bit data-memory page pointer field in ST0 ( $0 \leq DP \leq 511$ )

Table 1–1. Instruction Set Symbols and Abbreviations (Continued)

Symbol	Meaning
dst	Destination accumulator (A or B)
dst_	Opposite destination accumulator: If dst = A, then dst_ = B If dst = B, then dst_ = A
EAB	E address bus
EAR	EAB address register
extpmad	23-bit immediate program-memory address
FRCT	Fractional mode bit in ST1
hi(A)	High part of accumulator A (bits 32–16)
HM	Hold mode bit in ST1
IFR	Interrupt flag register
INTM	Interrupt mask bit in ST1
K	Short-immediate value of less than 9 bits
k3	3-bit immediate value ( $0 \leq k3 \leq 7$ )
k5	5-bit immediate value ( $-16 \leq k5 \leq 15$ )
k9	9-bit immediate value ( $0 \leq k9 \leq 511$ )
lk	16-bit long-immediate value
Lmem	32-bit single data-memory operand using long-word addressing
mmr, MMR	Memory-mapped register
MMRx, MMRy	Memory-mapped register, AR0–AR7 or SP
n	Number of words following the XC instruction; $n = 1$ or $2$
N	Designates the status register modified in the RSBX, SSBX, and XC instructions: N = 0 Status register ST0 N = 1 Status register ST1
OVA	Overflow flag for accumulator A in ST0
OVB	Overflow flag for accumulator B in ST0

*Table 1–1. Instruction Set Symbols and Abbreviations (Continued)*

Symbol	Meaning
OVdst	Overflow flag for the destination accumulator (A or B)
OVdst_	Overflow flag for the opposite destination accumulator (A or B)
OVsrc	Overflow flag for the source accumulator (A or B)
OVM	Overflow mode bit in ST1
PA	16-bit port immediate address ( $0 \leq PA \leq 65\,535$ )
PAR	Program address register
PC	Program counter
pmad	16-bit immediate program-memory address ( $0 \leq pmad \leq 65\,535$ )
Pmem	Program-memory operand
PMST	Processor mode status register
prog	Program-memory operand
[R]	Rounding option
rnd	Round
RC	Repeat counter
RTN	Fast-return register used in RETF[D] instruction
REA	Block-repeat end address register
RSA	Block-repeat start address register
SBIT	4-bit value that designates the status register bit number modified in the RSBX, SSBX, and XC instructions ( $0 \leq SBIT \leq 15$ )
SHFT	4-bit shift value ( $0 \leq SHFT \leq 15$ )
SHIFT	5-bit shift value ( $-16 \leq SHIFT \leq 15$ )
Sind	Single data-memory operand using indirect addressing
Smem	16-bit single data-memory operand
SP	Stack pointer
src	Source accumulator (A or B)
ST0, ST1	Status register 0, status register 1

Table 1–1. Instruction Set Symbols and Abbreviations (Continued)

Symbol	Meaning
SXM	Sign-extension mode bit in ST1
T	Temporary register
TC	Test/control flag in ST0
TOS	Top of stack
TRN	Transition register
TS	Shift value specified by bits 5–0 of T ( $-16 \leq TS \leq 31$ )
uns	Unsigned
XF	External flag status bit in ST1
XPC	Program counter extension register
Xmem	16-bit dual data-memory operand used in dual-operand instructions and some single-operand instructions
Ymem	16-bit dual data-memory operand used in dual-operand instructions

Table 1–2. Opcode Symbols and Abbreviations

Symbol	Meaning
A	Data-memory address bit
ARX	3-bit value that designates the auxiliary register
BITC	4-bit bit code
CC	2-bit condition code
CCCC CCCC	8-bit condition code
COND	4-bit condition code
D	Destination (dst) accumulator bit D = 0 Accumulator A D = 1 Accumulator B
I	Addressing mode bit I = 0 Direct addressing mode I = 1 Indirect addressing mode

Table 1–2. Opcode Symbols and Abbreviations (Continued)

Symbol	Meaning
K	Short-immediate value of less than 9 bits
MMRX	4-bit value that designates one of nine memory-mapped registers ( $0 \leq \text{MMRX} \leq 8$ )
MMRY	4-bit value that designates one of nine memory-mapped registers ( $0 \leq \text{MMRY} \leq 8$ )
N	Single bit
NN	2-bit value that determines the type of interrupt
R	Rounding (rnd) option bit R = 0 Execute instruction without rounding R = 1 Round the result
S	Source (src) accumulator bit S = 0 Accumulator A S = 1 Accumulator B
SBIT	4-bit status register bit number
SHFT	4-bit shift value ( $0 \leq \text{SHFT} \leq 15$ )
SHIFT	5-bit shift value ( $-16 \leq \text{SHIFT} \leq 15$ )
X	Data-memory bit
Y	Data-memory bit
Z	Delay instruction bit Z = 0 Execute instruction without delay Z = 1 Execute instruction with delay

Table 1–3. Instruction Set Notations

Symbol	Meaning
<b>Boldface Characters</b>	Boldface characters in an instruction syntax must be typed as shown. <i>Example:</i> For the syntax <b>ADD</b> <i>Xmem</i> , <i>Ymem</i> , <i>dst</i> , you can use a variety of values for <i>Xmem</i> and <i>Ymem</i> , but the word <b>ADD</b> must be typed as shown.
<i>italic symbols</i>	Italic symbols in an instruction syntax represent variables. <i>Example:</i> For the syntax <b>ADD</b> <i>Xmem</i> , <i>Ymem</i> , <i>dst</i> , you can use a variety of values for <i>Xmem</i> and <i>Ymem</i> .
[ x ]	Operands in square brackets are optional. <i>Example:</i> For the syntax <b>ADD</b> <i>Smem</i> [, <i>SHIFT</i> ], <i>src</i> [, <i>dst</i> ], you must use a value for <i>Smem</i> and <i>src</i> ; however, <i>SHIFT</i> and <i>dst</i> are optional.
#	Prefix of constants used in immediate addressing. For short- or long-immediate operands, # is used in instructions where there is ambiguity with other addressing modes that use immediate operands. For example:  RPT #15 uses short immediate addressing. It causes the next instruction to be repeated 16 times.  RPT 15 uses direct addressing. The number of times the next instruction repeats is determined by a value stored in memory.  For instructions using immediate operands for which there is no ambiguity, # is accepted by the assembler. For example, RPTZ A, #15 and RPTZ A, 15 are equivalent.
(abc)	The content of a register or location abc. <i>Example:</i> (src) means <i>the content of the source accumulator</i> .
x → y	Value x is assigned to register or location y. <i>Example:</i> (Smem) → dst means <i>the content of the data-memory value is loaded into the destination accumulator</i> .
r(n–m)	Bits n through m of register or location r. <i>Example:</i> src(15–0) means <i>bits 15 through 0 of the source accumulator</i> .
<< nn	Shift of nn bits left (negative or positive)
	Parallel instruction
\	Rotate left
//	Rotate right
$\bar{x}$	Logical inversion (1s complement) of x
x	Absolute value of x
AAh	Indicates that AA represents a hexadecimal number

Table 1–4. Operators Used in Instruction Set

Symbols	Operators	Evaluation
+   –   ~	Unary plus, minus, 1s complement	Right to left
*   /   %	Multiplication, division, modulo	Left to right
+   –	Addition, subtraction	Left to right
<<   >>	Left shift, right shift	Left to right
<   ≤	Less than, LT or equal	Left to right
>   ≥	Greater than, GT or equal	Left to right
≠   !=	Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
	Bitwise OR	Left to right

**Note:** Unary +, –, and \* have higher precedence than the binary forms.



## **1.2 Example Description of Instruction**

This example of a typical instruction description is provided to familiarize you with the format of the instruction descriptions and to explain what is described under each heading. Each instruction description in Chapter 4 presents the following information:

- ☐ Assembler syntax
- ☐ Operands
- ☐ Opcode
- ☐ Execution
- ☐ Status Bits
- ☐ Description
- ☐ Words
- ☐ Cycles
- ☐ Classes
- ☐ Examples

Each instruction description begins with an assembly syntax expression. Labels may be placed either before the instruction on the same line or on the preceding line in the first column. An optional comment field may conclude the syntax expression. Spaces are required between the fields:

- ☐ Label
- ☐ Command and operands
- ☐ Comment

## Example Description of Instruction

**Syntax**

- 1: **EXAMPLE** *Smem*, *src*
- 2: **EXAMPLE** *Smem*, **TS**, *src*
- 3: **EXAMPLE** *Smem*, **16**, *src* [, *dst*]
- 4: **EXAMPLE** *Smem* [, *SHIFT*], *src* [, *dst*]

Each instruction description begins with an assembly syntax expression. See Section 1.1 on page 1-2 for definitions of symbols in the syntax.

**Operands**

*Smem*: Single data-memory operand  
*Xmem*, *Ymem*: Dual data-memory operands  
*src*, *dst*: A (accumulator A)  
B (accumulator B)

$-16 \leq \text{SHIFT} \leq 15$

Operands may be constants or assembly-time expressions that refer to memory, I/O ports, register addresses, pointers, and a variety of other constants. This section also gives the range of acceptable values for the operand types.

## Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

The opcode breaks down the various bit fields that make up each instruction. See Section 1.1 on page 1-2 for definitions of symbols in the instruction opcode.

**Execution**

- 1: (*Smem*) + (*src*) → *src*
- 2: (*Smem*) << (**TS**) + (*src*) → *src*
- 3: (*Smem*) << **16** + (*src*) → *dst*
- 4: (*Smem*) [ << *SHIFT* ] + (*src*) → *dst*

The execution section describes the processing that takes place when the instruction is executed. The example executions are numbered to correspond to the numbered syntaxes. See Section 1.1 on page 1-2 for definitions of symbols in the execution.

**Status Bits**

An instruction's execution may be affected by the state of the fields in the status registers; also it may affect the state of the status register fields. Both the effects *on* and the effects *of* the status register fields are listed in this section.

**Description**

This section describes the instruction execution and its effect on the rest of the processor or on memory contents. Any constraints on the operands imposed by the processor or the assembler are discussed. The description parallels and supplements the information given symbolically in the execution section.

### *Example Description of Instruction*

<b>Words</b>	This field specifies the number of memory words required to store the instruction and its extension words. For instructions operating in single-addressing mode, the number of words given is for all modifiers except for long-offset modifiers, which require one additional word.
<b>Cycles</b>	This field specifies the number of cycles required for a given '54x instruction to execute as a single instruction with data accesses in DARAM and program accesses from ROM. Additional details on the number of cycles required for other memory configurations and repeat modes are given in Chapter 3, <i>Instruction Classes and Cycles</i> .
<b>Classes</b>	This field specifies the instruction class for each syntax of the instruction. See Chapter 3, <i>Instruction Classes and Cycles</i> , for a description of each class.
<b>Example</b>	Example code is included for each instruction. The effect of the code on memory and/or registers is summarized when appropriate.

# Instruction Set Summary

---

The '54x instruction set can be divided into four basic types of operations:

- ☐ Arithmetic operations
- ☐ Logical operations
- ☐ Program-control operations
- ☐ Load and store operations

In this chapter, each of the types of operations is divided into smaller groups of instructions with similar functions. With each instruction listing, you will find the best possible numbers for word count and cycle time, and the instruction class. You will also find a page number that directs you to the appropriate place in the instruction set of Chapter 4. Also included is information on repeating a single instruction and a list of nonrepeatable instructions.

Topic	Page
2.1 Arithmetic Operations .....	2-2
2.2 Logical Operations .....	2-8
2.3 Program-Control Operations .....	2-10
2.4 Load and Store Operations .....	2-14
2.5 Repeating a Single Instruction .....	2-19

## 2.1 Arithmetic Operations

This section summarizes the arithmetic operation instructions. Table 2–1 through Table 2–6 list the instructions within the following functional groups:

- ☐ Add instructions (Table 2–1)
- ☐ Subtract instructions (Table 2–2 on page 2-3)
- ☐ Multiply instructions (Table 2–3 on page 2-4)
- ☐ Multiply-accumulate instructions (Table 2–4 on page 2-4)
- ☐ Multiply-subtract instructions (Table 2–4 on page 2-4)
- ☐ Double (32-Bit operand) instructions (Table 2–5 on page 2-6)
- ☐ Application-specific instructions (Table 2–6 on page 2-7)

Table 2–1. Add Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ADD <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} + \text{Smem}$	1	1	3A, 3B	4-4
ADD <i>Smem</i> , TS, <i>src</i>	$\text{src} = \text{src} + \text{Smem} \ll \text{TS}$	1	1	3A, 3B	4-4
ADD <i>Smem</i> , 16, <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} + \text{Smem} \ll 16$	1	1	3A, 3B	4-4
ADD <i>Smem</i> [, SHIFT], <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} + \text{Smem} \ll \text{SHIFT}$	2	2	4A, 4B	4-4
ADD <i>Xmem</i> , SHFT, <i>src</i>	$\text{src} = \text{src} + \text{Xmem} \ll \text{SHFT}$	1	1	3A	4-4
ADD <i>Xmem</i> , <i>Ymem</i> , <i>dst</i>	$\text{dst} = \text{Xmem} \ll 16 + \text{Ymem} \ll 16$	1	1	7	4-4
ADD # <i>lk</i> [, SHFT], <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} + \#lk \ll \text{SHFT}$	2	2	2	4-4
ADD # <i>lk</i> , 16, <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} + \#lk \ll 16$	2	2	2	4-4
ADD <i>src</i> [, SHIFT] [, <i>dst</i> ]	$\text{dst} = \text{dst} + \text{src} \ll \text{SHIFT}$	1	1	1	4-4
ADD <i>src</i> , ASM [, <i>dst</i> ]	$\text{dst} = \text{dst} + \text{src} \ll \text{ASM}$	1	1	1	4-4
ADDC <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} + \text{Smem} + \text{C}$	1	1	3A, 3B	4-8
ADDM # <i>lk</i> , <i>Smem</i>	$\text{Smem} = \text{Smem} + \#lk$	2	2	18A, 18B	4-9
ADDS <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} + \text{uns}(\text{Smem})$	1	1	3A, 3B	4-10

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–2. Subtract Instructions

Syntax	Expression	W <sup>†</sup>	Cycles <sup>†</sup>	Class	Page
SUB <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} - \text{Smem}$	1	1	3A, 3B	4-187
SUB <i>Smem</i> , TS, <i>src</i>	$\text{src} = \text{src} - \text{Smem} \ll \text{TS}$	1	1	3A, 3B	4-187
SUB <i>Smem</i> , 16, <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} - \text{Smem} \ll 16$	1	1	3A, 3B	4-187
SUB <i>Smem</i> [, SHIFT], <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} - \text{Smem} \ll \text{SHIFT}$	2	2	4A, 4B	4-187
SUB <i>Xmem</i> , SHFT, <i>src</i>	$\text{src} = \text{src} - \text{Xmem} \ll \text{SHFT}$	1	1	3A	4-187
SUB <i>Xmem</i> , <i>Ymem</i> , <i>dst</i>	$\text{dst} = \text{Xmem} \ll 16 - \text{Ymem} \ll 16$	1	1	7	4-187
SUB # <i>lk</i> [, SHFT], <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} - \#lk \ll \text{SHFT}$	2	2	2	4-187
SUB # <i>lk</i> , 16, <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} - \#lk \ll 16$	2	2	2	4-187
SUB <i>src</i> [, SHIFT] [, <i>dst</i> ]	$\text{dst} = \text{dst} - \text{src} \ll \text{SHIFT}$	1	1	1	4-187
SUB <i>src</i> , ASM [, <i>dst</i> ]	$\text{dst} = \text{dst} - \text{src} \ll \text{ASM}$	1	1	1	4-187
SUBB <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} - \text{Smem} - \overline{\text{C}}$	1	1	3A, 3B	4-191
SUBC <i>Smem</i> , <i>src</i>	If $(\text{src} - \text{Smem} \ll 15) \geq 0$ $\text{src} = (\text{src} - \text{Smem} \ll 15) \ll 1 + 1$ Else $\text{src} = \text{src} \ll 1$	1	1	3A, 3B	4-192
SUBS <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} - \text{uns}(\text{Smem})$	1	1	3A, 3B	4-194

<sup>†</sup> Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–3. Multiply Instructions

Syntax	Expression	W†	Cycles†	Class	Page
MPY <i>Smem, dst</i>	$dst = T * Smem$	1	1	3A, 3B	4-101
MPYR <i>Smem, dst</i>	$dst = rnd(T * Smem)$	1	1	3A, 3B	4-101
MPY <i>Xmem, Ymem, dst</i>	$dst = Xmem * Ymem, T = Xmem$	1	1	7	4-101
MPY <i>Smem, #lk, dst</i>	$dst = Smem * \#lk, T = Smem$	2	2	6A, 6B	4-101
MPY <i>#lk, dst</i>	$dst = T * \#lk$	2	2	2	4-101
MPYA <i>dst</i>	$dst = T * A(32-16)$	1	1	1	4-104
MPYA <i>Smem</i>	$B = Smem * A(32-16), T = Smem$	1	1	3A, 3B	4-104
MPYU <i>Smem, dst</i>	$dst = uns(T) * uns(Smem)$	1	1	3A, 3B	4-106
SQUR <i>Smem, dst</i>	$dst = Smem * Smem, T = Smem$	1	1	3A, 3B	4-161
SQUR <i>A, dst</i>	$dst = A(32-16) * A(32-16)$	1	1	1	4-161

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–4. Multiply-Accumulate and Multiply-Subtract Instructions †

Syntax	Expression	W†	Cycles†	Class	Page
MAC <i>Smem, src</i>	$src = src + T * Smem$	1	1	3A, 3B	4-82
MAC <i>Xmem, Ymem, src [, dst]</i>	$dst = src + Xmem * Ymem,$ $T = Xmem$	1	1	7	4-82
MAC <i>#lk, src [, dst]</i>	$dst = src + T * \#lk$	2	2	2	4-82
MAC <i>Smem, #lk, src [, dst]</i>	$dst = src + Smem * \#lk,$ $T = Smem$	2	2	6A, 6B	4-82
MACR <i>Smem, src</i>	$dst = rnd(src + T * Smem)$	1	1	3A, 3B	4-82
MACR <i>Xmem, Ymem, src [, dst]</i>	$dst = rnd(src + Xmem * Ymem),$ $T = Xmem$	1	1	7	4-82
MACA <i>Smem [, B]</i>	$B = B + Smem * A(32-16),$ $T = Smem$	1	1	3A, 3B	4-85
MACA <i>T, src [, dst]</i>	$dst = src + T * A(32-16)$	1	1	1	4-85

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–4. Multiply-Accumulate and Multiply-Subtract Instructions (Continued)<sup>t</sup>

Syntax	Expression	W <sup>†</sup>	Cycles <sup>†</sup>	Class	Page
MACAR <i>Smem</i> [ , <i>B</i> ] <sup>t</sup>	$B = \text{md}(B + \text{Smem} * A(32-16)),$ $T = \text{Smem}$	1	1	3A, 3B	4-85
MACAR <i>T</i> , <i>src</i> [ , <i>dst</i> ]	$\text{dst} = \text{rnd}(\text{src} + T * A(32-16))$	1	1	1	4-85
MACD <i>Smem</i> , <i>pmad</i> , <i>src</i>	$\text{src} = \text{src} + \text{Smem} * \text{pmad},$ $T = \text{Smem}, (\text{Smem} + 1) = \text{Smem}$	2	3	23A, 23B	4-87
MACP <i>Smem</i> , <i>pmad</i> , <i>src</i>	$\text{src} = \text{src} + \text{Smem} * \text{pmad},$ $T = \text{Smem}$	2	3	22A, 22B	4-89
MACSU <i>Xmem</i> , <i>Ymem</i> , <i>src</i>	$\text{src} = \text{src} + \text{uns}(\text{Xmem}) * \text{Ymem},$ $T = \text{Xmem}$	1	1	7	4-91
MAS <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} - T * \text{Smem}$	1	1	3A, 3B	4-94
MASR <i>Xmem</i> , <i>Ymem</i> , <i>src</i> [ , <i>dst</i> ]	$\text{dst} = \text{rnd}(\text{src} - \text{Xmem} * \text{Ymem}),$ $T = \text{Xmem}$	1	1	7	4-94
MAS <i>Xmem</i> , <i>Ymem</i> , <i>src</i> [ , <i>dst</i> ]	$\text{dst} = \text{src} - \text{Xmem} * \text{Ymem},$ $T = \text{Xmem}$	1	1	7	4-94
MASR <i>Smem</i> , <i>src</i>	$\text{src} = \text{rnd}(\text{src} - T * \text{Smem})$	1	1	3A, 3B	4-94
MASA <i>Smem</i> [ , <i>B</i> ]	$B = B - \text{Smem} * A(32-16),$ $T = \text{Smem}$	1	1	3A, 3B	4-97
MASA <i>T</i> , <i>src</i> [ , <i>dst</i> ]	$\text{dst} = \text{src} - T * A(32-16)$	1	1	1	4-97
MASAR <i>T</i> , <i>src</i> [ , <i>dst</i> ]	$\text{dst} = \text{md}(\text{src} - T * A(32-16))$	1	1	1	4-97
SQURA <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} + \text{Smem} * \text{Smem},$ $T = \text{Smem}$	1	1	3A, 3B	4-163
SQURS <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} - \text{Smem} * \text{Smem},$ $T = \text{Smem}$	1	1	3A, 3B	4-164

<sup>†</sup> Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.



Table 2–5. Double (32-Bit Operand) Instructions

Syntax	Expression	W <sup>†</sup>	Cycles <sup>†</sup>	Class	Page
DADD <i>Lmem, src [ , dst]</i>	If C16 = 0 dst = Lmem + src If C16 = 1 dst(39–16) = Lmem(31–16) + src(31–16) dst(15–0) = Lmem(15–0) + src(15–0)	1	1	9A, 9B	4-37
DADST <i>Lmem, dst</i>	If C16 = 0 dst = Lmem + (T << 16 + T) If C16 = 1 dst(39–16) = Lmem(31–16) + T dst(15–0) = Lmem(15–0) + T	1	1	9A, 9B	4-39
DRSUB <i>Lmem, src</i>	If C16 = 0 src = Lmem – src If C16 = 1 src(39–16) = Lmem(31–16) – src(31–16) src(15–0) = Lmem(15–0) – src(15–0)	1	1	9A, 9B	4-43
DSADT <i>Lmem, dst</i>	If C16 = 0 dst = Lmem – (T << 16 + T) If C16 = 1 dst(39–16) = Lmem(31–16) – T dst(15–0) = Lmem(15–0) – T	1	1	9A, 9B	4-45
DSUB <i>Lmem, src</i>	If C16 = 0 src = src – Lmem If C16 = 1 src(39–16) = src(31–16) – Lmem(31–16) src(15–0) = src(15–0) – Lmem(15–0)	1	1	9A, 9B	4-48
DSUBT <i>Lmem, dst</i>	If C16 = 0 dst = Lmem – (T << 16 + T) If C16 = 1 dst(39–16) = Lmem(31–16) – T dst(15–0) = Lmem(15–0) – T	1	1	9A, 9B	4-50

<sup>†</sup> Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–6. Application-Specific Instructions

Syntax	Expression	W <sup>†</sup>	Cycles <sup>†</sup>	Class	Page
ABDST <i>Xmem</i> , <i>Ymem</i>	$B = B +  A(32-16) $ $A = (Xmem - Ymem) \ll 16$	1	1	7	4-2
ABS <i>src</i> [, <i>dst</i> ]	$dst =  src $	1	1	1	4-3
CMPL <i>src</i> [, <i>dst</i> ]	$dst = \sim src$	1	1	1	4-32
DELAY <i>Smem</i>	$(Smem + 1) = Smem$	1	1	24A, 24B	4-41
EXP <i>src</i>	$T = \text{number of sign bits}(src) - 8$	1	1	1	4-52
FIRS <i>Xmem</i> , <i>Ymem</i> , <i>pmad</i>	$B = B + A * pmad$ $A = (Xmem + Ymem) \ll 16$	2	3	8	4-59
LMS <i>Xmem</i> , <i>Ymem</i>	$B = B + Xmem * Ymem$ $A = (A + Xmem \ll 16) + 2^{15}$	1	1	7	4-80
MAX <i>dst</i>	$dst = \max(A, B)$	1	1	1	4-99
MIN <i>dst</i>	$dst = \min(A, B)$	1	1	1	4-100
NEG <i>src</i> [, <i>dst</i> ]	$dst = -src$	1	1	1	4-119
NORM <i>src</i> [, <i>dst</i> ]	$dst = src \ll TS$ $dst = \text{norm}(src, TS)$	1	1	1	4-122
POLY <i>Smem</i>	$B = Smem \ll 16$ $A = md(A * T + B)$	1	1	3A, 3B	4-126
RND <i>src</i> [, <i>dst</i> ]	$dst = src + 2^{15}$	1	1	1	4-142
SAT <i>src</i>	$\text{saturate}(src)$	1	1	1	4-154
SQDST <i>Xmem</i> , <i>Ymem</i>	$B = B + A(32-16) * A(32-16)$ $A = (Xmem + Ymem) \ll 16$	1	1	7	4-160

<sup>†</sup> Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

## 2.2 Logical Operations

This section summarizes the logical operation instructions. Table 2–7 through Table 2–11 list the instructions within the following functional groups:

- ☐ AND instructions (Table 2–7)
- ☐ OR instructions (Table 2–8 on page 2-8)
- ☐ XOR instructions (Table 2–9 on page 2-9)
- ☐ Shift instructions (Table 2–10 on page 2-9)
- ☐ Test instructions (Table 2–11 on page 2-9)

Table 2–7. AND Instructions

Syntax	Expression	W†	Cycles†	Class	Page
AND <i>Smem</i> , <i>src</i>	<i>src</i> = <i>src</i> & <i>Smem</i>	1	1	3A, 3B	4-11
AND <i>#lk</i> [ , <i>SHFT</i> ], <i>src</i> [ , <i>dst</i> ]	<i>dst</i> = <i>src</i> & <i>#lk</i> << <i>SHFT</i>	2	2	2	4-11
AND <i>#lk</i> , 16, <i>src</i> [ , <i>dst</i> ]	<i>dst</i> = <i>src</i> & <i>#lk</i> << 16	2	2	2	4-11
AND <i>src</i> [ , <i>SHIFT</i> ] [ , <i>dst</i> ]	<i>dst</i> = <i>dst</i> & <i>src</i> << <i>SHIFT</i>	1	1	1	4-11
ANDM <i>#lk</i> , <i>Smem</i>	<i>Smem</i> = <i>Smem</i> & <i>#lk</i>	2	2	18A, 18B	4-13

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–8. OR Instructions

Syntax	Expression	W†	Cycles†	Class	Page
OR <i>Smem</i> , <i>src</i>	<i>src</i> = <i>src</i>   <i>Smem</i>	1	1	3A, 3B	4-123
OR <i>#lk</i> [ , <i>SHFT</i> ], <i>src</i> [ , <i>dst</i> ]	<i>dst</i> = <i>src</i>   <i>#lk</i> << <i>SHFT</i>	2	2	2	4-123
OR <i>#lk</i> , 16, <i>src</i> [ , <i>dst</i> ]	<i>dst</i> = <i>src</i>   <i>#lk</i> << 16	2	2	2	4-123
OR <i>src</i> [ , <i>SHIFT</i> ] [ , <i>dst</i> ]	<i>dst</i> = <i>dst</i>   <i>src</i> << <i>SHIFT</i>	1	1	1	4-123
ORM <i>#lk</i> , <i>Smem</i>	<i>Smem</i> = <i>Smem</i>   <i>#lk</i>	2	2	18A, 18B	4-125

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–9. XOR Instructions

Syntax	Expression	W†	Cycles†	Class	Page
XOR <i>Smem</i> , <i>src</i>	$\text{src} = \text{src} \wedge \text{Smem}$	1	1	3A, 3B	4-201
XOR <i>#lk</i> [, <i>SHIFT</i> ], <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} \wedge \#lk \ll \text{SHIFT}$	2	2	2	4-201
XOR <i>#lk</i> , 16, <i>src</i> [, <i>dst</i> ]	$\text{dst} = \text{src} \wedge \#lk \ll 16$	2	2	2	4-201
XOR <i>src</i> [, <i>SHIFT</i> ] [, <i>dst</i> ]	$\text{dst} = \text{dst} \wedge \text{src} \ll \text{SHIFT}$	1	1	1	4-201
XORM <i>#lk</i> , <i>Smem</i>	$\text{Smem} = \text{Smem} \wedge \#lk$	2	2	18A, 18B	4-203

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–10. Shift Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ROL <i>src</i>	Rotate left with carry in	1	1	1	4-143
ROLTC <i>src</i>	Rotate left with TC in	1	1	1	4-144
ROR <i>src</i>	Rotate right with carry in	1	1	1	4-145
SFTA <i>src</i> , <i>SHIFT</i> [, <i>dst</i> ]	$\text{dst} = \text{src} \ll \text{SHIFT}$ {arithmetic shift}	1	1	1	4-155
SFTC <i>src</i>	if $\text{src}(31) = \text{src}(30)$ then $\text{src} = \text{src} \ll 1$	1	1	1	4-157
SFTL <i>src</i> , <i>SHIFT</i> [, <i>dst</i> ]	$\text{dst} = \text{src} \ll \text{SHIFT}$ {logical shift}	1	1	1	4-158

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–11. Test Instructions

Syntax	Expression	W†	Cycles†	Class	Page
BIT <i>Xmem</i> , <i>BITC</i>	$\text{TC} = \text{Xmem}(15 - \text{BITC})$	1	1	3A	4-21
BITF <i>Smem</i> , <i>#lk</i>	$\text{TC} = (\text{Smem} \&\& \#lk)$	2	2	6A, 6B	4-22
BITT <i>Smem</i>	$\text{TC} = \text{Smem}(15 - \text{T}(3-0))$	1	1	3A, 3B	4-23
CMPM <i>Smem</i> , <i>#lk</i>	$\text{TC} = (\text{Smem} == \#lk)$	2	2	6A, 6B	4-33
CMPR <i>CC</i> , <i>ARx</i>	Compare <i>ARx</i> with <i>AR0</i>	1	1	1	4-34

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

## 2.3 Program-Control Operations

This section summarizes the program-control instructions. Table 2–12 through Table 2–18 list the instructions within the following functional groups:

- ☐ Branch instructions (Table 2–12)
- ☐ Call instructions (Table 2–13 on page 2-11)
- ☐ Interrupt instructions (Table 2–14 on page 2-11)
- ☐ Return instructions (Table 2–15 on page 2-12)
- ☐ Repeat instructions (Table 2–16 on page 2-12)
- ☐ Stack-manipulating instructions (Table 2–17 on page 2-13)
- ☐ Miscellaneous program-control instructions (Table 2–18 on page 2-13)

Table 2–12. Branch Instructions

Syntax	Expression	W†	Cycles†	Class	Page
B[D] <i>pmad</i>	PC = pmad(15–0)	2	4/[2¶]	29A	4-14
BACC[D] <i>src</i>	PC = src(15–0)	1	6/[4¶]	30A	4-15
BANZ[D] <i>pmad, Sind</i>	if (Sind ≠ 0) then PC = pmad(15–0)	2	4‡/2§/[2¶]	29A	4-16
BC[D] <i>pmad, cond [ , cond [ , cond ]]</i>	if (cond(s)) then PC = pmad(15–0)	2	5‡/3§/[3¶]	31A	4-18
FB[D] <i>extpmad</i>	PC = pmad(15–0), XPC = pmad(22–16)	2	4/[2¶]	29A	4-53
FBACC[D] <i>src</i>	PC = src(15–0), XPC = src(22–16)	1	6/[4¶]	30A	4-54

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

‡ Conditions true

§ Condition false

¶ Delayed instruction

Table 2–13. Call Instructions

Syntax	Expression	W†	Cycles†	Class	Page
CALA[D] <i>src</i>	--SP = PC, PC = src(15–0)	1	6/[4‡]	30B	4-25
CALL[D] <i>pmad</i>	--SP = PC, PC = pmad(15–0)	2	4/[2§]	29B	4-27
CC[D] <i>pmad, cond [ , cond [ , cond ]]</i>	if (cond(s)) then --SP = PC, PC = pmad(15–0)	2	5‡/3§/ [3‡]	31B	4-29
FCALA[D] <i>src</i>	--SP = PC, --SP = XPC, PC = src(15–0), XPC = src(22–16)	1	6/[4‡]	30B	4-55
FCALL[D] <i>extpmad</i>	--SP = PC, --SP = XPC, PC = pmad(15–0), XPC = pmad(22–16)	2	4/[2‡]	29B	4-57

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

‡ Conditions true

§ Condition false

‡ Delayed instruction

Table 2–14. Interrupt Instructions

Syntax	Expression	W†	Cycles†	Class	Page
INTR <i>K</i>	--SP = PC, PC = IPTR(15–7) + K << 2, INTM = 1	1	3	35	4-65
TRAP <i>K</i>	--SP = PC, PC = IPTR(15–7) + K << 2	1	3	35	4-195

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–15. Return Instructions

Syntax	Expression	W†	Cycles†	Class	Page
FRET[D]	XPC = SP++, PC = SP++	1	6/[4‡]	34	4-61
FRETE[D]	XPC = SP++, PC = SP++, INTM = 0	1	6/[4‡]	34	4-62
RC[D] <i>cond</i> [ , <i>cond</i> [ , <i>cond</i> ]]	if ( <i>cond</i> (s)) then PC = SP++	1	5‡/3§/[3‡]	32	4-133
RET[D]	PC = SP++	1	5/[3‡]	32	4-139
RETE[D]	PC = SP++, INTM = 0	1	5/[3‡]	32	4-140
RETF[D]	PC = RTN, PC++, INTM = 0	1	3/[1‡]	33	4-141

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

‡ Conditions true

§ Condition false

‡ Delayed instruction

Table 2–16. Repeat Instructions

Syntax	Expression	W†	Cycles†	Class	Page
RPT <i>Smem</i>	Repeat single, RC = <i>Smem</i>	1	1	5A, 5B	4-146
RPT #K	Repeat single, RC = #K	1	1	1	4-146
RPT #lk	Repeat single, RC = #lk	2	2	2	4-146
RPTB[D] <i>pmad</i>	Repeat block, RSA = PC + 2[4#], REA = <i>pmad</i> – 1	2	4/[2‡]	29A	4-148
RPTZ <i>dst</i> , #lk	Repeat single, RC = #lk, <i>dst</i> = 0	2	2	2	4-150

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

‡ Delayed instruction

Table 2–17. Stack-Manipulating Instructions

Syntax	Expression	W†	Cycles†	Class	Page
FRAME <i>K</i>	$SP = SP + K$	1	1	1	4-60
POPD <i>Smem</i>	$Smem = SP++$	1	1	17A, 17B	4-127
POPM <i>MMR</i>	$MMR = SP++$	1	1	17A	4-128
PSHD <i>Smem</i>	$--SP = Smem$	1	1	16A, 16B	4-131
PSHM <i>MMR</i>	$--SP = MMR$	1	1	16A	4-132

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–18. Miscellaneous Program-Control Instructions

Syntax	Expression	W†	Cycles†	Class	Page
IDLE <i>K</i>	idle( <i>K</i> )	1	4	36	4-63
MAR <i>Smem</i>	If CMPT = 0, then modify ARx If CMPT = 1 and ARx $\neq$ AR0, then modify ARx, ARP = x If CMPT = 1 and ARx = AR0, then modify AR(ARP)	1	1	1, 2	4-92
NOP	no operation	1	1	1	4-121
RESET	software reset	1	3	35	4-138
RSBX <i>N, SBIT</i>	STN (SBIT) = 0	1	1	1	4-151
SSBX <i>N, SBIT</i>	STN (SBIT) = 1	1	1	1	4-166
XC <i>n, cond</i> [ , <i>cond</i> [ , <i>cond</i> ]]	If (cond(s)) then execute the next <i>n</i> instructions; <i>n</i> = 1 or 2	1	1	1	4-198

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.



## 2.4 Load and Store Operations

This section summarizes the load and store instructions. Table 2–19 through Table 2–26 list the instructions within the following functional groups:

- ☐ Load instructions (Table 2–19)
- ☐ Store instructions (Table 2–20 on page 2-15)
- ☐ Conditional store instructions (Table 2–21 on page 2-16)
- ☐ Parallel load and store instructions (Table 2–22 on page 2-16)
- ☐ Parallel load and multiply instructions (Table 2–23 on page 2-16)
- ☐ Parallel store and add/subtract instructions (Table 2–24 on page 2-17)
- ☐ Parallel store and multiply instructions (Table 2–25 on page 2-17)
- ☐ Miscellaneous load-type and store-type instructions (Table 2–26 on page 2-18)

Table 2–19. Load Instructions

Syntax	Expression	W†	Cycles†	Class	Page
DLD <i>Lmem, dst</i>	<i>dst</i> = <i>Lmem</i>	1	1	9A, 9B	4-42
LD <i>Smem, dst</i>	<i>dst</i> = <i>Smem</i>	1	1	3A, 3B	4-66
LD <i>Smem, TS, dst</i>	<i>dst</i> = <i>Smem</i> << TS	1	1	3A, 3B	4-66
LD <i>Smem, 16, dst</i>	<i>dst</i> = <i>Smem</i> << 16	1	1	3A, 3B	4-66
LD <i>Smem</i> [ , <i>SHIFT</i> ], <i>dst</i>	<i>dst</i> = <i>Smem</i> << SHIFT	2	2	4A, 4B	4-66
LD <i>Xmem, SHFT, dst</i>	<i>dst</i> = <i>Xmem</i> << SHFT	1	1	3A	4-66
LD # <i>K, dst</i>	<i>dst</i> = # <i>K</i>	1	1	1	4-66
LD # <i>lk</i> [ , <i>SHFT</i> ], <i>dst</i>	<i>dst</i> = # <i>lk</i> << SHFT	2	2	2	4-66
LD # <i>lk, 16, dst</i>	<i>dst</i> = # <i>lk</i> << 16	2	2	2	4-66
LD <i>src, ASM</i> [ , <i>dst</i> ]	<i>dst</i> = <i>src</i> << ASM	1	1	1	4-66
LD <i>src</i> [ , <i>SHIFT</i> ] [ , <i>dst</i> ]	<i>dst</i> = <i>src</i> << SHIFT	1	1	1	4-66
LD <i>Smem, T</i>	<i>T</i> = <i>Smem</i>	1	1	3A, 3B	4-70
LD <i>Smem, DP</i>	<i>DP</i> = <i>Smem</i> (8–0)	1	3	5A, 5B	4-70
LD # <i>k9, DP</i>	<i>DP</i> = # <i>k9</i>	1	1	1	4-70
LD # <i>k5, ASM</i>	<i>ASM</i> = # <i>k5</i>	1	1	1	4-70

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–19. Load Instructions (Continued)

Syntax	Expression	W†	Cycles†	Class	Page
LD #k3, ARP	ARP = #k3	1	1	1	4-70
LD Smem, ASM	ASM = Smem(4–0)	1	1	3A, 3B	4-70
LDM MMR, dst	dst = MMR	1	1	3A	4-73
LDR Smem, dst	dst = rnd(Smem)	1	1	3A, 3B	4-78
LDU Smem, dst	dst = uns(Smem)	1	1	3A, 3B	4-79
LTD Smem	T = Smem, (Smem + 1) = Smem	1	1	24A, 24B	4-81

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–20. Store Instructions

Syntax	Expression	W†	Cycles†	Class	Page
DST src, Lmem	Lmem = src	1	2	13A, 13B	4-47
ST T, Smem	Smem = T	1	1	10A, 10B	4-167
ST TRN, Smem	Smem = TRN	1	1	10A, 10B	4-167
ST #lk, Smem	Smem = #lk	2	2	12A, 12B	4-167
STH src, Smem	Smem = src << –16	1	1	10A, 10B	4-169
STH src, ASM, Smem	Smem = src << (ASM – 16)	1	1	10A, 10B	4-169
STH src, SHFT, Xmem	Xmem = src << (SHFT – 16)	1	1	10A	4-169
STH src[, SHIFT], Smem	Smem = src << (SHIFT – 16)	2	2	11A, 11B	4-169
STL src, Smem	Smem = src	1	1	10A, 10B	4-172
STL src, ASM, Smem	Smem = src << ASM	1	1	10A, 10B	4-172
STL src, SHFT, Xmem	Xmem = src << SHFT	1	1	10A, 10B	4-172
STL src[, SHIFT], Smem	Smem = src << SHIFT	2	2	11A, 11B	4-172
STLM src, MMR	MMR = src	1	1	10A	4-175
STM #lk, MMR	MMR = #lk	2	2	12A	4-176

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–21. Conditional Store Instructions

Syntax	Expression	W†	Cycles†	Class	Page
CMPS <i>src, Smem</i>	If $\text{src}(31-16) > \text{src}(15-0)$ then $\text{Smem} = \text{src}(31-16)$ If $\text{src}(31-16) \leq \text{src}(15-0)$ then $\text{Smem} = \text{src}(15-0)$	1	1	10A, 10B	4-35
SACCD <i>src, Xmem, cond</i>	If (cond) $\text{Xmem} = \text{src} << (\text{ASM} - 16)$	1	1	15	4-152
SRCCD <i>Xmem, cond</i>	If (cond) $\text{Xmem} = \text{BRC}$	1	1	15	4-165
STRCD <i>Xmem, cond</i>	If (cond) $\text{Xmem} = \text{T}$	1	1	15	4-186

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–22. Parallel Load and Store Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ST <i>src, Ymem</i>    LD <i>Xmem, dst</i>	$\text{Ymem} = \text{src} << (\text{ASM} - 16)$    $\text{dst} = \text{Xmem} << 16$	1	1	14	4-178
ST <i>src, Ymem</i>    LD <i>Xmem, T</i>	$\text{Ymem} = \text{src} << (\text{ASM} - 16)$    $\text{T} = \text{Xmem}$	1	1	14	4-178

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–23. Parallel Load and Multiply Instructions

Syntax	Expression	W†	Cycles†	Class	Page
LD <i>Xmem, dst</i>    MAC <i>Ymem, dst_</i>	$\text{dst} = \text{Xmem} << 16$    $\text{dst}_ = \text{dst}_ + \text{T} * \text{Ymem}$	1	1	7	4-74
LD <i>Xmem, dst</i>    MACR <i>Ymem, dst_</i>	$\text{dst} = \text{Xmem} << 16$    $\text{dst}_ = \text{rnd}(\text{dst}_ + \text{T} * \text{Ymem})$	1	1	7	4-74
LD <i>Xmem, dst</i>    MAS <i>Ymem, dst_</i>	$\text{dst} = \text{Xmem} << 16$    $\text{dst}_ = \text{dst}_ - \text{T} * \text{Ymem}$	1	1	7	4-76
LD <i>Xmem, dst</i>    MASR <i>Ymem, dst_</i>	$\text{dst} = \text{Xmem} << 16$    $\text{dst}_ = \text{rnd}(\text{dst}_ - \text{T} * \text{Ymem})$	1	1	7	4-76

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–24. Parallel Store and Add/Subtract Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ST <i>src</i> , <i>Ymem</i>    ADD <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = dst\_ + Xmem \ll 16$	1	1	14	4-177
ST <i>src</i> , <i>Ymem</i>    SUB <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = (Xmem \ll 16) - dst\_$	1	1	14	4-185

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–25. Parallel Store and Multiply Instructions

Syntax	Expression	W†	Cycles†	Class	Page
ST <i>src</i> , <i>Ymem</i>    MAC <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = dst + T * Xmem$	1	1	14	4-180
ST <i>src</i> , <i>Ymem</i>    MACR <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = rnd(dst + T * Xmem)$	1	1	14	4-180
ST <i>src</i> , <i>Ymem</i>    MAS <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = dst - T * Xmem$	1	1	14	4-182
ST <i>src</i> , <i>Ymem</i>    MASR <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = rnd(dst - T * Xmem)$	1	1	14	4-182
ST <i>src</i> , <i>Ymem</i>    MPY <i>Xmem</i> , <i>dst</i>	$Ymem = src \ll (ASM - 16)$ $  \ dst = T * Xmem$	1	1	14	4-184

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

Table 2–26. Miscellaneous Load-Type and Store-Type Instructions

Syntax	Expression	W†	Cycles†	Class	Page
MVDD <i>Xmem, Ymem</i>	<i>Ymem</i> = <i>Xmem</i>	1	1	14	4-107
MVDK <i>Smem, dmad</i>	<i>dmad</i> = <i>Smem</i>	2	2	19A, 19B	4-108
MVDM <i>dmad, MMR</i>	<i>MMR</i> = <i>dmad</i>	2	2	19A	4-110
MVDP <i>Smem, pmad</i>	<i>pmad</i> = <i>Smem</i>	2	4	20A, 20B	4-111
MVKD <i>dmad, Smem</i>	<i>Smem</i> = <i>dmad</i>	2	2	19A, 19B	4-113
MVMD <i>MMR, dmad</i>	<i>dmad</i> = <i>MMR</i>	2	2	19A	4-115
MVMM <i>MMRx, MMRy</i>	<i>MMRy</i> = <i>MMRx</i>	1	1	1	4-116
MVPD <i>pmad, Smem</i>	<i>Smem</i> = <i>pmad</i>	2	3	21A, 21B	4-117
PORTR <i>PA, Smem</i>	<i>Smem</i> = <i>PA</i>	2	2	27A, 27B	4-129
PORTW <i>Smem, PA</i>	<i>PA</i> = <i>Smem</i>	2	2	28A, 28B	4-130
READA <i>Smem</i>	<i>Smem</i> = <i>A</i>	1	5	25A, 25B	4-136
WRITA <i>Smem</i>	<i>A</i> = <i>Smem</i>	1	5	26A, 26B	4-196

† Values for words (W) and cycles assume the use of DARAM for data. Add 1 word and 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

## 2.5 Repeating a Single Instruction

The '54x includes repeat instructions that cause the next instruction to be repeated. The number of times for the instruction to be repeated is obtained from an operand of the instruction and is equal to this operand + 1. This value is stored in the 16-bit repeat counter (RC) register. You cannot program the value in the RC register; it is loaded by the repeat instructions only. The maximum number of executions of a given instruction is 65 536. An absolute program or data address is automatically incremented when the single-repeat feature is used.

Once a repeat instruction is decoded, all interrupts, including  $\overline{\text{NMI}}$  but not  $\overline{\text{RS}}$ , are disabled until the completion of the repeat loop. However, the '54x does respond to the HOLD signal while executing a repeat loop—the response depends on the value of the HM bit of status register 1 (ST1).

The repeat function can be used with some instructions, such as multiply/accumulate and block moves, to increase the execution speed of these instructions. These multicycle instructions (Table 2–27) effectively become single-cycle instructions after the first iteration of a repeat instruction.

*Table 2–27. Multicycle Instructions That Become Single-Cycle Instructions When Repeated*

Instruction	Description	# Cycles†
FIRS	Symmetrical FIR filter	3
MACD	Multiply and move result in accumulator with delay	3
MACP	Multiply and move result in accumulator	3
MVDK	Data-to-data move	2
MVDM	Data-to-MMR move	2
MVDP	Data-to-program move	4
MVKD	Data-to-data move	2
MVMD	MMR-to-data move	2
MVPD	Program-to-data move	3
READA	Read from program-memory to data memory	5
WRITA	Write data memory to program memory	5

† Number of cycles when instruction is not repeated

### Repeating a Single Instruction

Single data-memory operand instructions cannot be repeated if a long offset modifier or an absolute address is used (for example, \*ARn(lk), \*+ARn(lk), \*+ARn(lk)% and \*(lk)). Instructions listed in Table 2–28 cannot be repeated using RPT or RPTZ instructions.

Table 2–28. Nonrepeatable Instructions

Instruction	Description
ADDM	Add long constant to data memory
ANDM	AND data memory with long constant
B[D]	Unconditional branch
BACC[D]	Branch to accumulator address
BANZ[D]	Branch on auxiliary register not 0
BC[D]	Conditional branch
CALA[D]	Call to accumulator address
CALL[D]	Unconditional call
CC[D]	Conditional call
CMPR	Compare with auxiliary register
DST	Long word (32-bit) store
FB[D]	Far branch unconditionally
FBACC[D]	Far branch to location specified by accumulator
FCALA[D]	Far call subroutine at location specified by accumulator
FCALL[D]	Far call unconditionally
FRET[D]	Far return
FRETE[D]	Enable interrupts and far return from interrupt
IDLE	Idle instructions
INTR	Interrupt trap
LD ARP	Load auxiliary register pointer (ARP)
LD DP	Load data page pointer (DP)
MVMM	Move memory-mapped register (MMR) to another MMR
ORM	OR data memory with long constant

Table 2–28. Nonrepeatable Instructions (Continued)

Instruction	Description
RC[D]	Conditional return
RESET	Software reset
RET[D]	Unconditional return
RETE[D]	Return from interrupt
RETF[D]	Fast return from interrupt
RND	Round accumulator
RPT	Repeat next instruction
RPTB[D]	Block repeat
RPTZ	Repeat next instruction and clear accumulator
RSBX	Reset status register bit
SSBX	Set status register bit
TRAP	Software trap
XC	Conditional execute
XORM	XOR data memory with long constant



## Instruction Classes and Cycles

---

Instructions are classified into several categories, or classes, according to cycles required. This chapter describes the instruction classes. Because a single instruction can have multiple syntaxes and types of execution, it can appear in multiple classes.

The tables in this chapter show the number of cycles required for a given '54x instruction to execute in a given memory configuration when executed as a single instruction and when executed in the repeat mode. Tables are also provided for a single data-memory operand access used with a long constant. The column headings in the tables indicate the program source location. These headings are defined as follows:

- ROM** The instruction executes from internal program ROM.
- SARAM** The instruction executes from internal single-access RAM.
- DARAM** The instruction executes from internal dual-access RAM.
- External** The instruction executes from external program memory.

If a class of instructions requires memory operand(s), the row divisions in the tables indicate the location(s) of the operand(s). These locations are defined as follows:

- DARAM** The operand is in internal dual-access RAM.
- SARAM** The operand is in internal single-access RAM.
- DROM** The operand is in internal data ROM.
- PROM** The operand is in internal program ROM.
- External** The operand is in external memory.
- MMR** The operand is a memory-mapped register.

The number of cycles required for each instruction is given in terms of the processor machine cycles (the CLKOUT period). The additional wait states for program/data memory accesses and I/O accesses are defined as follows:

- d** Data-memory wait states—the number of additional clock cycles the device waits for external data-memory to respond to an access.

- io** I/O wait states—the number of additional clock cycles the device waits for an external I/O to respond to an access.
- n** Repetitions—the number of times a repeated instruction is executed (where  $n > 2$  to fill the pipeline).
- nd** Data-memory wait states repeated  $n$  times.
- np** Program-memory wait states repeated  $n$  times.
- npd** Program-memory wait states repeated  $n$  times.
- p** Program-memory wait states—the number of additional clock cycles the device waits for external program memory to respond to an access.
- pd** Program-memory wait states—the number of additional clock cycles the device waits for external program memory to respond to an access as a program data operand.

These variables can also use the subscripts *src*, *dst*, and *code* to indicate source, destination, and code, respectively.

All reads from external memory take at least one instruction cycle to complete, and all writes to external memory take at least two instruction cycles to complete. These external accesses take longer if additional wait-state cycles are added using the software wait-state generator or the external READY input. However, internal to the CPU all writes to external memory take only one cycle as long as no other access to the external memory is in process at the same time. This is possible because the instruction pipeline takes only one cycle to request an external write access, and the external bus interface unit completes the write access independently.

The instruction cycles are based on the following assumptions:

- ☐ At least five instructions following the current instruction are fetched from the same memory section (internal or external) as the current instruction, except in instructions that cause a program counter (PC) discontinuity, such as a branch or call.
- ☐ When executing a single instruction, there is no pipeline or bus conflict between the current instruction and any other instruction in the pipeline. The only exception is the conflict between the instruction fetch and the memory read/write access (if any) of the instruction under consideration.
- ☐ In single-instruction repeat mode, all conflicts caused by the pipelined execution of that instruction are considered.

**Class 1** 1 word, 1 cycle. No operand, or short-immediate or register operands and no memory operands.

**Mnemonics**

ABS	MACA[R]	NORM	SFTA
ADD	MAR	OR	SFTC
AND	MASA[R]	RND	SFTL
CMPL	MAX	ROL	SQUR
CMPR	MIN	ROLTC	SSBX
EXP	MPYA	ROR	SUB
FRAME	MVMM	RPT	XC
LD	NEG	RSBX	XOR
LD T/DP/ASM/ARP	NOP	SAT	

**Cycles****Cycles for a Single Execution**

Program		
ROM/SARAM	DARAM	External
1	1	1+p

**Cycles for a Repeat Execution**

Program		
ROM/SARAM	DARAM	External
n	n	n+p

## Class 2

**Class 2** 2 words, 2 cycles. Long-immediate operand and no memory operands.

<b>Mnemonics</b>	ADD	MAC	OR	SUB
	AND	MAR	RPT	XOR
	LD	MPY	RPTZ	

## Cycles

Cycles for a Single Execution		
Program		
ROM/SARAM	DARAM	External
2	2	2+2p

Cycles for a Repeat Execution		
Program		
ROM/SARAM	DARAM	External
n+1	n+1	n+1+2p

**Class 3A** 1 word, 1 cycle. Single data-memory (Smem or Xmem) read operand or MMR read operand.

**Mnemonics**

ADD	LDM	MPYA	SUBB
ADDC	LDR	MPYU	SUBC
ADDS	LDU	OR	SUBS
AND	MAC[R]	POLY	XOR
BIT	MACA[R]	SQUR	
BITT	MAS[R]	SQURA	
LD	MASA	SQURS	
LD T/DP/ASM/ARP	MPY[R]	SUB	

**Cycles**

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
DARAM	1	1, 2 <sup>†</sup>	1+p
SARAM	1, 2 <sup>†</sup>	1	1+p
DROM	1, 2 <sup>†</sup>	1	1+p
External	1+d	1+d	2+d+p
MMR <sup>◇</sup>	1	1	1+p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
DARAM	n	n, n+1 <sup>†</sup>	n+p
SARAM	n, n+1 <sup>†</sup>	n	n+p
DROM	n, n+1 <sup>†</sup>	n	n+p
External	n+nd	n+nd	n+1+nd+p
MMR <sup>◇</sup>	n	n	n+p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

### Class 3B

**Class 3B** 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing.

<b>Mnemonics</b>	ADD	LDU	OR	SUBS
	ADDC	MAC[R]	POLY	XOR
	ADDS	MACA[R]	SQUR	
	AND	MAS[R]	SQURA	
	BITT	MASA	SQURS	
	LD	MPY[R]	SUB	
	LD T/DP/ASM/ARP	MPYA	SUBB	
	LDR	MPYU	SUBC	

### Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
	ROM/SARAM	DARAM	External
DARAM	2	2, 3 <sup>†</sup>	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
DROM	2, 3 <sup>†</sup>	2	2+2p
External	2+d	2+d	3+d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 4A** 2 words, 2 cycles. Single data-memory (Smem) read operand.

**Mnemonics**            ADD                    LD                    SUB

**Cycles**

Cycles for a Single Execution			
Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	2	2, 3 <sup>†</sup>	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
DROM	2, 3 <sup>†</sup>	2	2+2p
External	2+d	2+d	3+d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution			
Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	n+1	n+1, n+2 <sup>†</sup>	n+1+2p
SARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
DROM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
External	n+1+nd	n+1+nd	n+2+nd+2p
MMR <sup>◇</sup>	n+1	n+1	n+1+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

## Class 4B

**Class 4B** 3 words, 3 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing.

### Mnemonics

ADD

LD

SUB

### Cycles

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
	ROM/SARAM	DARAM	External
DARAM	3	3, 4 <sup>†</sup>	3+3p
SARAM	3, 4 <sup>†</sup>	3	3+3p
DROM	3, 4 <sup>†</sup>	3	3+3p
External	3+d	3+d	4+d+3p
MMR <sup>◇</sup>	3	3	3+3p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.



**Class 5A** 1 word, 3 cycles. Single data-memory (Smem) read operand (with DP destination for load instruction).

**Mnemonics** LD RPT

**Cycles**

Operand	Cycles for a Single Execution		
	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	3	3	3+p
SARAM	3	3	3+p
DROM	3	3	3+p
External	3+d	3+d	3+d+p
MMR <sup>◇</sup>	3	3	3+p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 5B** 2 words, 4 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing (with DP destination for load instruction).

**Mnemonics** LD RPT

**Cycles**

Operand	Cycles for a Single Execution With Long-Offset Modifier		
	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	4	4	4+2p
SARAM	4	4	4+2p
DROM	4	4	4+2p
External	4+d	4+d	4+d+2p
MMR <sup>◇</sup>	4	4	4+2p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 6A** 2 words, 2 cycles. Single data-memory (Smem) read operand and single long-immediate operand.

**Mnemonics** BITF CMPM MAC MPY

**Cycles****Cycles for a Single Execution**

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	2	2, 3 <sup>†</sup>	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
DROM	2, 3 <sup>†</sup>	2	2+2p
External	2+d	2+d	3+d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Cycles for a Repeat Execution**

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	n+1	n+1, n+2 <sup>†</sup>	n+1+2p
SARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
DROM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
External	n+1+nd	n+1+nd	n+2+nd+2p
MMR <sup>◇</sup>	n+1	n+1	n+1+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

**Class 6B** 3 words, 3 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single long-immediate operand.

**Mnemonics** BITF CMPM MAC MPY

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	3	3, 4 <sup>†</sup>	3+3p
SARAM	3, 4 <sup>†</sup>	3	3+3p
DROM	3, 4 <sup>†</sup>	3	3+3p
External	3+d	3+d	4+d+3p
MMR <sup>◇</sup>	3	3	3+3p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

## ***Class 7***

---

***Class 7***      1 word, 1 cycle. Dual data-memory (Xmem and Ymem) read operands.

***Mnemonics***

ABDST	LD  MAS[R]	MACSU	SQDST
ADD	LMS	MAS[R]	SUB
LD  MAC[R]	MAC[R]	MPY	

**Cycles**

Cycles for a Single Execution				
Operand		Program		
Xmem	Ymem	ROM/SARAM	DARAM	External
DARAM	DARAM	1	1, 2 <sup>†</sup>	1+p
	SARAM	1, 2 <sup>†</sup>	1, 2 <sup>†</sup>	1+p
	DROM	1, 2 <sup>†</sup>	1, 2 <sup>†</sup>	1+p
	External	1+d	1+d, 2	2+d+p
SARAM	DARAM	1, 2 <sup>†</sup>	1	1+p
	SARAM	1, 2 <sup>†</sup> , 3 <sup>‡</sup>	1, 2 <sup>†</sup>	1+p, 2 <sup>*</sup>
	DROM	1, 2 <sup>†</sup>	1	1+p
	External	1+d, 2	1+d	2+d+p
DROM	DARAM	1, 2 <sup>†</sup>	1	1+p
	SARAM	1, 2 <sup>†</sup>	1, 2 <sup>†</sup>	1+p, 2 <sup>*</sup>
	DROM	1, 2 <sup>†</sup> , 3 <sup>‡</sup>	1, 2 <sup>†</sup>	1+p, 2 <sup>*</sup>
	External	1+d, 2	1+d	2+d+p
External	DARAM	1+d	1+d	2+d+p
	SARAM	1+d, 2	1+d	2+d+p
	DROM	1+d, 2	1+d	2+d+p
	External	2+2d	2+2d	3+2d+p
MMR <sup>◇</sup>	DARAM	1	1	1+p
	SARAM	1, 2 <sup>†</sup>	1	1+p
	DROM	1, 2 <sup>†</sup>	1	1+p
	External	1+d	1+d	2+d+p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

|| One operand and code in same memory block when d = 0

<sup>\*</sup> Two operands in same memory block when p = 0

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution				
Operand		Program		
Xmem	Ymem	ROM/SARAM	DARAM	External
DARAM	DARAM	n	n, n+1 <sup>†</sup>	n+p
	SARAM	n, n+1 <sup>†</sup>	n, n+1 <sup>†</sup>	n+p
	DROM	n, n+1 <sup>†</sup>	n, n+1 <sup>†</sup>	n+p
	External	n+nd	n+nd, 1+n	n+1+nd+p
SARAM	DARAM	n, n+1 <sup>†</sup>	n	n+p
	SARAM	n, n+1 <sup>†</sup> , 2n <sup>#</sup> , 2n+1 <sup>‡</sup>	n, 2n <sup>#</sup>	n+p, 2n (p = 0) <sup>#</sup> , 2n-1+p (p ≥ 1) <sup>#</sup>
	DROM	n, n+1 <sup>†</sup>	n	n+p
	External	n+nd, n+1	n+nd	n+1+nd+p
DROM	DARAM	n, n+1 <sup>†</sup>	n	n+p
	SARAM	n, n+1 <sup>†</sup>	n	n+p
	DROM	n, n+1 <sup>†</sup> , 2n <sup>#</sup> , 2n+1 <sup>‡</sup>	n, 2n <sup>#</sup>	n+p, 2n (p = 0) <sup>#</sup> , 2n-1+p (p ≥ 1) <sup>#</sup>
	External	n+nd, n+1	n+nd	n+1+nd+p
External	DARAM	n+nd	n+nd	n+1+nd+p
	SARAM	n+nd, n+1	n+nd	n+1+nd+p
	DROM	n+nd, n+1	n+nd	n+1+nd+p
	External	2n+2nd	2n+2nd	2n+1+2nd+p
MMR <sup>◇</sup>	DARAM	n	n	n+p
	SARAM	n, n+1 <sup>†</sup>	n	n+p
	DROM	n, n+1 <sup>†</sup>	n	n+p
	External	n+nd	n+nd	n+1+nd+p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>#</sup> Two operands in same memory block

|| One operand and code in same memory block when d = 0

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

**Class 8** 2 words, 3 cycles. Dual data-memory (Xmem and Ymem) read operands and a single program-memory (pmad) operand.

**Mnemonics** FIRS

**Cycles**

Cycles for a Single Execution					
Operand			Program		
pmad	Xmem	Ymem	ROM/SARAM	DARAM	External
DARAM	DARAM	DARAM	3, 4 <sup>†</sup>	3, 4 <sup>†</sup>	3+2p, 4+2p <sup>†</sup>
		SARAM/ DROM	3, 4 <sup>†</sup>	3, 4 <sup>†</sup>	3+2p, 4+2p <sup>†</sup>
		External	3+d, 4+d <sup>†</sup>	3+d, 4+d <sup>†</sup>	3+d+2p, 4+d+2p <sup>†</sup>
	SARAM/ DROM	DARAM	3	3	3+2p
		SARAM/ DROM	3, 4 <sup>‡</sup>	3, 4 <sup>‡</sup>	3+2p, 4+2p <sup>‡</sup>
		External	3+d	3+d	3+d+2p
	External	DARAM	3+d	3+d	3+d+2p
		SARAM/ DROM	3+d	3+d	3+d+2p
		External	4+2d	4+2d	4+2d+2p
SARAM/ DROM	DARAM	DARAM	3	3	3+2p
		SARAM/ DROM	3, 4 <sup>§</sup>	3, 4 <sup>§</sup>	3+2p, 4+2p <sup>§</sup>
		External	3+d	3+d	3+d+2p

<sup>†</sup> Xmem and pmad in same memory block

<sup>‡</sup> Xmem and Ymem in same memory block

<sup>§</sup> Ymem and pmad in same memory block

<sup>¶</sup> Xmem, Ymem, and pmad in same memory block

Cycles for a Single Execution (Continued)

Operand			Program		
pmad	Xmem	Ymem	ROM/SARAM	DARAM	External
	SARAM/ DROM	DARAM	3, 4†	3, 4†	3+2p, 4+2p†
		SARAM/ DROM	3, 4†, 5‡	3, 4†, 5‡	3+2p, 4+2p†, 5+2p‡
		External	3+d, 4+d†	3+d, 4+d†	3+d+2p, 4+d+2p†
	External	DARAM	3+d	3+d	3+2p
		SARAM/ DROM	3+d, 4+d§	3+d, 4+d§	3+2p, 4+d+2p§
		External	4+2d	4+2d	4+2d+2p
External	DARAM	DARAM	3+pd	3+pd	3+pd+2p
		SARAM/ DROM	3+pd	3+pd	3+pd+2p
		External	4+pd+d	4+pd+d	4+pd+d+2p
	SARAM/ DROM	DARAM	3+pd	3+pd	3+pd+2p
		SARAM/ DROM	3+pd, 4+pd†	3+pd, 4+pd†	3+pd+2p, 4+pd+2p†
		External	4+pd+d	4+pd+d	4+pd+d+2p
	External	DARAM	4+pd+d	4+pd+d	4+pd+d+2p
		SARAM/ DROM	4+pd+d	4+pd+d	4+pd+d+2p
		External	5+pd+2d	5+pd+2d	5+pd+2d +2p

† Xmem and pmad in same memory block

‡ Xmem and Ymem in same memory block

§ Ymem and pmad in same memory block

‡ Xmem, Ymem, and pmad in same memory block



Cycles for a Repeat Execution

Operand			Program		
pmad	Xmem	Ymem	ROM/ SARAM	DARAM	External
DARAM	DARAM	DARAM	$n+2, 2n+2^\dagger$	$n+2, 2n+2^\dagger$	$n+2+2p, 2n+2+2p^\dagger$
		SARAM/ DROM	$n+2, 2n+2^\dagger$	$n+2, 2n+2^\dagger$	$n+2+2p, 2n+2+2p^\dagger$
		External	$n+2+nd, 2n+2+nd^\dagger$	$n+2+nd, 2n+2+nd^\dagger$	$n+2+nd+2p, 2n+2+nd+2p^\dagger$
	SARAM/ DROM	DARAM	$n+2$	$n+2$	$n+2+2p$
		SARAM/ DROM	$n+2, 2n+2^\ddagger$	$n+2, 2n+2^\ddagger$	$n+2+2p, 2n+2+2p^\ddagger$
		External	$n+2+nd$	$n+2+nd$	$n+2+nd+2p$
	External	DARAM	$n+2+nd$	$n+2+nd$	$n+2+nd+2p$
		SARAM/ DROM	$n+2+nd$	$n+2+nd$	$n+2+nd+2p$
		External	$2n+2+2nd$	$2n+2+2nd$	$2n+2+2nd+2p$
SARAM/ DROM	DARAM	DARAM	$n+2$	$n+2$	$n+2+2p$
		SARAM/ DROM	$n+2, 2n+2^\S$	$n+2, 2n+2^\S$	$n+2+2p, 2n+2+2p^\S$
		External	$n+2+nd$	$n+2+nd$	$n+2+nd+2p$

$^\dagger$  Xmem and pmad in same memory block

$^\ddagger$  Xmem and Ymem in same memory block

$^\S$  Ymem and pmad in same memory block

$^\P$  Xmem, Ymem, and pmad in same memory block

Cycles for a Repeat Execution (Continued)

Operand			Program		
pmad	Xmem	Ymem	ROM/ SARAM	DARAM	External
	SARAM/ DROM	DARAM	$n+2, 2n+2^\dagger$	$n+2, 2n+2^\dagger$	$n+2+2p, 2n+2+2p^\dagger$
		SARAM/ DROM	$n+2, 2n+2^\dagger, 3n+2^\ddagger$	$n+2, 2n+2^\dagger, 3n+2^\ddagger$	$n+2+2p, 2n+2+2p^\dagger, 3n+2+2p^\ddagger$
		External	$n+2+nd, 2n+2+nd^\dagger$	$n+2+nd, 2n+2+nd^\dagger$	$n+2+nd+2p, 2n+2+nd+2p^\dagger$
	External	DARAM	$n+2+nd$	$n+2+nd$	$n+2+nd$
		SARAM/ DROM	$n+2+nd, 2n+2+nd^\S$	$n+2+nd, 2n+2+nd^\S$	$n+2+nd+2p, 2n+2+nd+2p^\S$
		External	$2n+2+2nd$	$2n+2+2nd$	$2n+2+2nd+2p$
External	DARAM	DARAM	$n+2+npd$	$n+2+npd$	$n+2+npd+2p$
		SARAM/ DROM	$n+2+npd$	$n+2+npd$	$n+2+npd+2p$
		External	$2n+2+npd+nd$	$2n+2+npd+nd$	$2n+2+npd+nd+2p$
	SARAM/ DROM	DARAM	$n+2+npd$	$n+2+npd$	$n+2+npd+2p$
		SARAM/ DROM	$n+2+npd, 2n+2+npd^\ddagger$	$n+2+npd, 2n+2+npd^\ddagger$	$n+2+npd+2p, 2n+2+npd+2p^\ddagger$
		External	$2n+2+npd+nd$	$2n+2+npd+nd$	$2n+2+npd+nd+2p$

$^\dagger$  Xmem and pmad in same memory block

$^\ddagger$  Xmem and Ymem in same memory block

$^\S$  Ymem and pmad in same memory block

$^\ddagger$  Xmem, Ymem, and pmad in same memory block

Cycles for a Repeat Execution (Continued)

Operand			Program		
pmad	Xmem	Ymem	ROM/ SARAM	DARAM	External
	External	DARAM	$2n+2+npd+nd$	$2n+2+npd+nd$	$2n+2+npd+nd+2p$
		SARAM/ DROM	$2n+2+npd+nd$	$2n+2+npd+nd$	$2n+2+npd+nd+2p$
		External	$3n+2+npd+2nd$	$3n+2+npd+2nd$	$3n+2+npd+2nd+2p$

† Xmem and pmad in same memory block

‡ Xmem and Ymem in same memory block

§ Ymem and pmad in same memory block

¶ Xmem, Ymem, and pmad in same memory block

# Class 9A

**Class 9A** 1 word, 1 cycle. Single long-word data-memory (Lmem) read operand.

<b>Mnemonics</b>	DADD	DLD	DSADT	DSUBT
	DADST	DRSUB	DSUB	

## Cycles

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
DARAM	1	1, 2†	1+p
SARAM	1, 2†	1	1+p
DROM	1, 2†	1	1+p
External	2+2d	2+2d	3+2d+p

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
DARAM	n	n, n+1†	n+p
SARAM	n, n+1†	n	n+p
DROM	n, n+1†	n	n+p
External	2n+2nd	2n+2nd	1+2n+2nd+p

† Operand and code in same memory block

**Class 9B** 2 words, 2 cycles. Single long-word data-memory (Lmem) read operand using long-offset indirect addressing.

**Mnemonics**

DADD	DLD	DSADT	DSUBT
DADST	DRSUB	DSUB	

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier			
Operand	Program		
Lmem	ROM/SARAM	DARAM	External
DARAM	2	2, 3 <sup>†</sup>	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
DROM	2, 3 <sup>†</sup>	2	2+2p
External	3+2d	3+2d	4+2d+2p

<sup>†</sup> Operand and code in same memory block

## Class 10A

**Class 10A** 1 word, 1 cycle. Single data-memory (Smem or Xmem) write operand or an MMR write operand.

**Mnemonics**

CMPS	STH	STLM
ST	STL	

### Cycles

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	1	1	1+p
SARAM	1, 2†	1	1+p
External	1	1	4+d+p
MMR <sup>◇</sup>	1	1	1+p

† Operand and code in same memory block

◇ Add n cycles for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	n	n	n+p
SARAM	n, n+1†	n	n+p
External	2n-1+(n-1)d	2n-1+(n-1)d	2n+2+nd+p
MMR <sup>◇</sup>	n	n	n+p

† Operand and code in same memory block

◇ Add n cycles for peripheral memory-mapped access.

**Class 10B** 2 words, 2 cycles. Single data-memory (Smem or Xmem) write operand using long-offset indirect addressing.

**Mnemonics** CMPS ST STH STL

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier			
Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	2	2	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
External	2	2	5+d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 11A** 2 words, 2 cycles. Single data-memory (Smem) write operand.

**Mnemonics**

STH

STL

**Cycles****Cycles for a Single Execution**

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	2	2	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
External	2	2	5+d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Cycles for a Repeat Execution**

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	n+1	n+1	n+1+2p
SARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
External	2n+(n-1)d	2n+(n-1)d	2n+3+nd+2p
MMR <sup>◇</sup>	n+1	n+1	n+1+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.



**Class 11B** 3 words, 3 cycles. Single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics**            STH                    STL

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier			
Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	3	3	3+3p
SARAM	3, 4†	3	3+3p
External	3	3	6+d+3p
MMR◇	3	3	3+3p

† Operand and code in same memory block

◇ Add one cycle for peripheral memory-mapped access.

**Class 12A** 2 words, 2 cycles. Single data-memory (Smem) write operand or MMR write operand.

**Mnemonics** ST STM

**Cycles**

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	2	2	2+2p
SARAM	2, 3 <sup>†</sup>	2	2+2p
External	2	2	5+d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	2n	2n	2n+2p
SARAM	2n, 2n+1 <sup>†</sup>	2n	2n+2p
External	2n+(n-1)d	2n+(n-1)d	2n+3+nd+p
MMR <sup>◇</sup>	2n	2n	2n+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

**Class 12B** 3 words, 3 cycles. Single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics** ST

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier			
Operand	Program		
Smem	ROM/SARAM	DARAM	External
DARAM	3	3	3+3p
SARAM	3, 4 <sup>†</sup>	3	3+3p
External	3	3	6+d+3p
MMR <sup>◇</sup>	3	3	3+3p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 13A** 1 word, 2 cycles. Single long-word data-memory (Lmem) write operand.

**Mnemonics** DST

**Cycles**

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Lmem			
DARAM	2	2	2+p
SARAM	2, 4 <sup>†</sup>	2	2+p
External	3+d	3+d	8+2d+p
MMR <sup>◇</sup>	2	2	2+p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Lmem			
DARAM	2n	2n	2n+p
SARAM	2n, 2n+2 <sup>†</sup>	2n	2n+p
External	4n-1+(2n-1)d	4n-1+(2n-1)d	4n+4+2nd+p
MMR <sup>◇</sup>	2n	2n	2n+p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

**Class 13B** 2 words, 3 cycles. Single long-word data-memory (Lmem) write operand using long-offset indirect addressing.

**Mnemonics** DST

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier			
Operand	Program		
Lmem	ROM/SARAM	DARAM	External
DARAM	3	3	3+2p
SARAM	3, 5 <sup>†</sup>	3	3+2p
External	4+d	4+d	9+2d+2p
MMR <sup>◇</sup>	3	3	3+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

## Class 14

**Class 14** 1 word, 1 cycle. Dual data-memory (Xmem and Ymem) read and write operands.

**Mnemonics**

MVDD	ST  LD	ST  MAS[R]	ST  SUB
ST  ADD	ST  MAC[R]	ST  MPY	

## Cycles

Cycles for a Single Execution				
Operand		Program		
Xmem	Ymem	ROM/SARAM	DARAM	External
DARAM	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1, 2†	1+p
	External	1	1, 2†	4+d+p
SARAM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†, 3‡	1	1+p
	External	1, 2†	1	4+d+p
DROM	DARAM	1, 2†	1	1+p
	SARAM	1, 2†	1	1+p
	External	1, 2†	1	4+d+p
External	DARAM	1+d	1+d	2+d+p
	SARAM	1+d, 2+d†	1+d	2+d+p
	External	1+d	1+d	5+2d+p
MMR <sup>◇</sup>	DARAM	1	1, 2†	1+p
	SARAM	1, 2†	1	1+p
	External	1	1	4+d+p

† Operand and code in same memory block

‡ Two operands and code in same memory block

◇ Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution				
Operand		Program		
Xmem	Ymem	ROM/SARAM	DARAM	External
DARAM	DARAM	n	n, n+1 <sup>†</sup>	n+p
	SARAM	n, n+1 <sup>†</sup>	n, n+1 <sup>†</sup>	n+p
	External	2n-1+(n-1)d	2n-1+(n-1)d, 2n+(n-1)d <sup>†</sup>	2n+2+nd+p
SARAM	DARAM	n, n+1 <sup>†</sup>	n	n+p
	SARAM	n, n+1 <sup>†</sup> , 2n <sup>#</sup> , 2n+1 <sup>‡</sup>	n, 2n <sup>#</sup>	n+p, 2n+p <sup>#</sup>
	External	2n-1+(n-1)d, 2n+(n-1)d <sup>†</sup>	2n-1+(n-1)d, 2n+(n-1)d <sup>†</sup>	2n+2+nd+p
DROM	DARAM	n, n+1 <sup>†</sup>	n, n+1 <sup>†</sup>	n+p
	SARAM	n, n+1 <sup>†</sup>	n	n+p
	External	2n-1+(n-1)d, 2n+(n-1)d <sup>†</sup>	2n-1+(n-1)d	2n+2+nd+p
External	DARAM	n+nd	n+nd	n+1+nd+p
	SARAM	n+nd, n+1+nd <sup>†</sup>	n+nd	n+1+nd+p
	External	4n-3+(2n-1)d	4n-3+(2n-1)d	4n+1+2nd+p
MMR <sup>◇</sup>	DARAM	n	n, 2n <sup>†</sup>	n+p
	SARAM	n, n+1 <sup>†</sup>	n	n+p
	External	2n-1+(n-1)d	2n-1+(n-1)d	2n+2+nd+p

<sup>†</sup> Operand and code in same memory block  
<sup>‡</sup> Two operands and code in same memory block

<sup>#</sup> Two operands in same memory block  
<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

## Class 15

**Class 15** 1 word, 1 cycle. Single data-memory (Xmem) write operand.

**Mnemonics** SACCD SRCCD STRCD

### Cycles

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Xmem			
DARAM	1	1	1+p
SARAM	1, 2 <sup>†</sup>	1	1+p
External	1	1	4+d+p
MMR <sup>◇</sup>	1	1	1+p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Xmem			
DARAM	n	n	n+p
SARAM	n, n+1 <sup>†</sup>	n	n+p
External	2n-1+(n-1)d	2n-1+(n-1)d	2n+2+nd+p
MMR <sup>◇</sup>	n	n	n+p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.



**Class 16A** 1 word, 1 cycle. Single data-memory (Smem) read operand or MMR read operand, and a stack-memory write operand.

**Mnemonics** PSHD PSHM

**Cycles**

Cycles for a Single Execution				
Operand		Program		
Smem	Stack	ROM/SARAM	DARAM	External
DARAM	DARAM	1	1, 2 <sup>†</sup>	1+p
	SARAM	1, 2 <sup>†</sup>	1, 2 <sup>†</sup>	1+p
	External	1	1, 2 <sup>†</sup>	4+d+p
SARAM	DARAM	1, 2 <sup>†</sup>	1	1+p
	SARAM	1, 2 <sup>†</sup> , 3 <sup>‡</sup>	1	1+p
	External	1, 2 <sup>†</sup>	1	4+d+p
DROM	DARAM	1, 2 <sup>†</sup>	1	1+p
	SARAM	1, 2 <sup>†</sup>	1	1+p
	External	1, 2 <sup>†</sup>	1	4+d+p
External	DARAM	1+d	1+d	2+d+p
	SARAM	1+d, 2+d <sup>†</sup>	1+d	2+d+p
	External	1+d	1+d	5+2d+p
MMR <sup>◇</sup>	DARAM	1	1, 2 <sup>†</sup>	1+p
	SARAM	1, 2 <sup>†</sup>	1	1+p
	External	1	1	4+d+p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	Stack	ROM/SARAM	DARAM	External
DARAM	DARAM	n	n, n+1†	n+p
	SARAM	n, n+1†	n, n+1†	n+p
	External	2n-1+(n-1)d	2n-1+(n-1)d, 2n+(n-1)d†	2n+2+nd+p
SARAM	DARAM	n, n+1†	n	n+p
	SARAM	n, n+1†, 2n <sup>#</sup> , 2n+1‡	n, 2n <sup>#</sup>	n+p, 2n+p <sup>#</sup>
	External	2n-1+(n-1)d, 2n+(n-1)d†	2n-1+(n-1)d, 2n+(n-1)d†	2n+2+nd+p
DROM	DARAM	n, n+1†	n, n+1†	n+p
	SARAM	n, n+1†	n	n+p
	External	2n-1+(n-1)d, 2n+(n-1)d†	2n-1+(n-1)d	2n+2+nd+p
External	DARAM	n+nd	n+nd	n+1+nd+p
	SARAM	n+nd, n+1+nd†	n+nd	n+1+nd+p
	External	4n-3+(2n-1)d	4n-3+(2n-1)d	4n+1+2nd+p
MMR <sup>◇</sup>	DARAM	n	n, 2n†	n+p
	SARAM	n, n+1†	n	n+p
	External	2n-1+(n-1)d	2n-1+(n-1)d	2n+2+nd+p

† Operand and code in same memory block

‡ Two operands and code in same memory block

# Two operands in same memory block

◇ Add n cycles for peripheral memory-mapped access.

**Class 16B** 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and a stack-memory write operand.

**Mnemonics** PSHD

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
Smem	Stack	ROM/SARAM	DARAM	External
DARAM	DARAM	2	2, 3 <sup>†</sup>	2+2p
	SARAM	2, 3 <sup>†</sup>	2, 3 <sup>†</sup>	2+2p
	External	2	2, 3 <sup>†</sup>	5+d+2p
SARAM	DARAM	2, 3 <sup>†</sup>	2	2+2p
	SARAM	2, 3 <sup>†</sup> , 4 <sup>‡</sup>	2	2+2p
	External	2, 3 <sup>†</sup>	2	5+d+2p
DROM	DARAM	2, 3 <sup>†</sup>	2	2+2p
	SARAM	2, 3 <sup>†</sup>	2	2+2p
	External	2, 3 <sup>†</sup>	2	5+d+2p
External	DARAM	2+d	2+d	3+d+2p
	SARAM	2+d, 3+d <sup>†</sup>	2+d	3+d+2p
	External	2+d	2+d	6+2d+2p
MMR <sup>◇</sup>	DARAM	2	2, 3 <sup>†</sup>	2+2p
	SARAM	2, 3 <sup>†</sup>	2	2+2p
	External	2	2	5+d+2p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

## Class 17A

**Class 17A** 1 word, 1 cycle. Single data-memory (Smem) write operand or MMR write operand, and a stack-memory read operand.

**Mnemonics** POPD POPM

### Cycles

Cycles for a Single Execution				
Operand		Program		
Smem	Stack	ROM/SARAM	DARAM	External
DARAM	DARAM	1	1, 2 <sup>†</sup>	1+p
	SARAM	1, 2 <sup>†</sup>	1	1+p
	DROM	1, 2 <sup>†</sup>	1	1+p
	External	1+d	1+d	2+d+p
	MMR <sup>◇</sup>	1	1, 2 <sup>†</sup>	1+p
SARAM	DARAM	1, 2 <sup>†</sup>	1, 2 <sup>†</sup>	1+p
	SARAM	1, 2 <sup>†</sup> , 3 <sup>‡</sup>	1	1+p
	DROM	1, 2 <sup>†</sup>	1	1+p
	External	1+d, 2+d <sup>†</sup>	1+d	2+d+p
	MMR <sup>◇</sup>	1, 2 <sup>†</sup>	1	1+p
External	DARAM	1	1, 2 <sup>†</sup>	4+d+p
	SARAM	1, 2 <sup>†</sup>	1	4+d+p
	DROM	1, 2 <sup>†</sup>	1	4+d+p
	External	1+d	1+d	5+2d+p
	MMR <sup>◇</sup>	1	1	4+d+p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	Stack	ROM/SARAM	DARAM	External
DARAM	DARAM	n	n, n+1†	n+p
	SARAM	n, n+1†	n	n+p
	DROM	n, n+1†	n, n+1†	n+p
	External	n+nd	n+nd	n+1+nd+p
	MMR <sup>◇</sup>	n	n, 2n†	n+p
SARAM	DARAM	n, n+1†	n, n+1†	n+p
	SARAM	n, n+1†, 2n 2n+1†	n, 2n	n+p, 2n+p
	DROM	n, n+1†	n	n+p
	External	n+nd, n+1+nd†	n+nd	n+1+nd+p
	MMR <sup>◇</sup>	n, n+1†	n	n+p
External	DARAM	2n-1+(n-1)d	2n-1+(n-1)d, 2n+(n-1)d†	2n+2+nd+p
	SARAM	2n-1+(n-1)d, 2n+(n-1)d†	2n-1+(n-1)d, 2n+(n-1)d†	2n+2+nd+p
	DROM	2n-1+(n-1)d, 2n+(n-1)d†	2n-1+(n-1)d	2n+2+nd+p
	External	4n-3+((2n-1)d	4n-3+(2n-1)d	4n+1+2nd+p
	MMR <sup>◇</sup>	2n-1+(n-1)d	2n-1+(n-1)d	2n+2+nd+p

† Operand and code in same memory block

‡ Two operands and code in same memory block

◇ Add one cycle for peripheral memory-mapped access.

**Class 17B** 2 words, 2 cycles. Single data-memory (Smem) write operand using long-offset indirect addressing, and a stack-memory read operand.

**Mnemonics** POPD

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
Smem	Stack	ROM/SARAM	DARAM	External
DARAM	DARAM	2	2, 3 <sup>†</sup>	2+2p
	SARAM	2, 3 <sup>†</sup>	2	2+2p
	DROM	2, 3 <sup>†</sup>	2	2+2p
	External	2+d	2+d	3+d+2p
	MMR <sup>◇</sup>	2	2, 3 <sup>†</sup>	2+2p
SARAM	DARAM	2, 3 <sup>†</sup>	2, 3 <sup>†</sup>	2+2p
	SARAM	2, 3 <sup>†</sup> , 4 <sup>‡</sup>	2	2+2p
	DROM	2, 3 <sup>†</sup>	2	2+2p
	External	2+d, 3+d <sup>†</sup>	2+d	3+d+2p
	MMR <sup>◇</sup>	2, 3 <sup>†</sup>	2	2+2p
External	DARAM	2	2, 3 <sup>†</sup>	5+d+2p
	SARAM	2, 3 <sup>†</sup>	2	5+d+2p
	DROM	2, 3 <sup>†</sup>	2	5+d+2p
	External	2+d	2+d	6+2d+2p
	MMR <sup>◇</sup>	2	2	5+d+2p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 18A** 2 words, 2 cycles. Single data-memory (Smem) read and write operand.

**Mnemonics**            ADDM                    ANDM                    ORM                    XORM

**Cycles**

Operand	Cycles for a Single Execution		
	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	2	2, 3 <sup>†</sup>	2+2p
SARAM	2, 4 <sup>†</sup>	2	2+2p
External	2+d	2+d	6+2d+2p
MMR <sup>◇</sup>	2	2	2+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 18B** 3 words, 3 cycles. Single data-memory (Smem) read and write operand using long-offset indirect addressing.

**Mnemonics**            ADDM                    ANDM                    ORM                    XORM

**Cycles**

Operand	Cycles for a Single Execution With Long-Offset Modifier		
	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	3	3, 4 <sup>†</sup>	3+3p
SARAM	3, 5 <sup>†</sup>	3	3+3p
External	3+d	3+d	7+2d+3p
MMR <sup>◇</sup>	3	3	3+3p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

## Class 19A

**Class 19A** 2 words, 2 cycles. Single data-memory (Smem) read operand or MMR read operand, and single data-memory (dmem) write operand; or single data-memory (dmem) read operand, and single data-memory (Smem) write operand or MMR write operand.

### Mnemonics

MVDK

MVDM

MVKD

MVMD

### Cycles

Cycles for a Single Execution

Operand		Program		
Smem	dmem	ROM/SARAM	DARAM	External
DARAM	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2, 3†	2+2p
	External	2	2, 3†	5+d+2p
	MMR <sup>◇</sup>	2	2	2+2p
SARAM	DARAM	2, 3†	2	2+2p
	SARAM	2, 3†, 4†	2	2+2p
	External	2, 3†	2	5+d+2p
	MMR <sup>◇</sup>	2, 3†	2	2+2p
DROM	DARAM	2, 3†	2	2+2p
	SARAM	2, 3†	2	2+2p
	External	2, 3†	2	5+d+2p
	MMR <sup>◇</sup>	2, 3†	2	2+2p
External	DARAM	2+d	2+d	3+d+2p
	SARAM	2+d, 3+d†	2+d	3+d+2p
	External	2+d	2+d	6+2d+p
	MMR <sup>◇</sup>	2+d	2+d	3+d+2p
MMR <sup>◇</sup>	DARAM	2	2, 3†	2+2p
	SARAM	2, 3†	2	2+2p
	External	2	2	5+d+2p
	MMR <sup>◇</sup>	2	2	2+2p

† Operand and code in same memory block

‡ Two operands and code in same memory block

◇ Add one cycle for peripheral memory-mapped access.



Cycles for a Repeat Execution

Operand		Program		
Smem	dmad	ROM/SARAM	DARAM	External
DARAM	DARAM	n+1	n+1, n+2 <sup>†</sup>	n+1+2p
	SARAM	n+1, n+2 <sup>†</sup>	n+1, n+2 <sup>†</sup>	n+1+2p
	External	2n+(n-1)d	2n+(n-1)d, 2n+1+(n-1)d <sup>†</sup>	2n+3+nd+2p
	MMR <sup>◇</sup>	n+1	n+1	n+1+2p
SARAM	DARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
	SARAM	2n, 2n+1 <sup>†</sup> , 2n+2 <sup>‡</sup>	2n	2n+2p
	External	2n+(n-1)d, 2n+1+(n-1)d <sup>†</sup>	2n+(n-1)d	2n+3+nd+2p
	MMR <sup>◇</sup>	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
DROM	DARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
	SARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
	External	2n+(n-1)d, 2n+1+(n-1)d <sup>†</sup>	2n+(n-1)d	2n+3+nd+2p
	MMR <sup>◇</sup>	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
External	DARAM	n+1+nd	n+1+nd	n+1+nd+2p
	SARAM	n+1+nd, n+2nd <sup>†</sup>	n+1+nd	n+1+nd+2p
	External	4n-2+(2n-1)d	4n-2+(2n-1)d	4n+2+2nd+2p
	MMR <sup>◇</sup>	n+1+nd	n+1+nd	n+1+nd+2p
MMR <sup>◇</sup>	DARAM	n+1	n+1	n+1+2p
	SARAM	n+1, n+2 <sup>†</sup>	n+1	n+1+2p
	External	2n+(n-1)d	2n+(n-1)d	2n+3+nd+2p
	MMR <sup>◇</sup>	n+1	n+1	n+1+2p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

**Class 19B** 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single data-memory (dmad) write operand, or single data-memory (dmad) read operand and single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics**

MVDK

MVKD

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	dmad	ROM/SARAM	DARAM	External
DARAM	DARAM	3	3, 4 <sup>†</sup>	3+3p
	SARAM	3, 4 <sup>†</sup>	3, 4 <sup>†</sup>	3+3p
	External	3	3, 4 <sup>†</sup>	6+d+3p
	MMR <sup>◇</sup>	3	3	3+3p
SARAM	DARAM	3, 4 <sup>†</sup>	3	3+3p
	SARAM	3, 4 <sup>†</sup> , 5 <sup>‡</sup>	3	3+3p
	External	3, 4 <sup>†</sup>	3	6+d+3p
	MMR <sup>◇</sup>	3, 4 <sup>†</sup>	3	3+3p
DROM	DARAM	3, 4 <sup>†</sup>	3	3+3p
	SARAM	3, 4 <sup>†</sup>	3	3+3p
	External	3, 4 <sup>†</sup>	3	6+d+3p
	MMR <sup>◇</sup>	3, 4 <sup>†</sup>	3	3+3p
External	DARAM	3+d	3+d	4+d+3p
	SARAM	3+d, 4+d <sup>†</sup>	3+d	4+d+3p
	External	3+d	3+d	7+2d+2p
	MMR <sup>◇</sup>	3+d	3+d	4+d+3p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Single Execution With Long-Offset Modifier (Continued)

Operand		Program		
Smem	dmad	ROM/SARAM	DARAM	External
MMR <sup>◇</sup>	DARAM	3	3, 4 <sup>†</sup>	3+3p
	SARAM	3, 4 <sup>†</sup>	3	3+3p
	External	3	3	6+d+3p
	MMR <sup>◇</sup>	3	3	3+3p

<sup>†</sup> Operand and code in same memory block

<sup>‡</sup> Two operands and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

## Class 20A

**Class 20A** 2 words, 4 cycles. Single data-memory (Smem) read operand and single program-memory (pmad) write operand.

**Mnemonics** MVDP

### Cycles

Cycles for a Single Execution				
Operand		Program		
Smem	pmad	ROM/SARAM	DARAM	External
DARAM	DARAM	4	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p
SARAM	DARAM	4, 5 <sup>†</sup>	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p
DROM	DARAM	4, 5 <sup>†</sup>	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p
External	DARAM	4+d	4+d	4+d+2p
	SARAM	4+d	4+d	4+d+2p
	External	4+d+pd	4+d+pd	6+d+pd+2p
MMR <sup>◇</sup>	DARAM	4	4	4+2p
	SARAM	4	4	4+2p
	External	4	4	6+pd+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	pmad	ROM/SARAM	DARAM	External
DARAM	DARAM	$n+3$	$n+3$	$n+3+2p$
	SARAM	$n+3$	$n+3$	$n+3+2p$
	External	$2n+2+(n-1)pd$	$2n+2+(n-1)pd$	$2n+4+npd+2p$
SARAM	DARAM	$n+3$	$n+3$	$n+3+2p$
	SARAM	$n+3, 2n+2^{\#}$	$n+3, 2n+2^{\#}$	$n+3+2p, 2n+2+2p^{\#}$
	External	$2n+2+(n-1)pd$	$2n+2+(n-1)pd$	$2n+4+npd+2p$
DROM	DARAM	$n+3$	$n+3$	$n+3+2p$
	SARAM	$n+3$	$n+3$	$n+3+2p$
	External	$2n+2+(n-1)pd$	$2n+2+(n-1)pd$	$2n+4+npd+2p$
External	DARAM	$n+3+npd$	$n+3+npd$	$n+3+npd+2p$
	SARAM	$n+3+npd$	$n+3+npd$	$n+3+npd+2p$
	External	$4n+nd+npd$	$4n+nd+npd$	$4n+2+nd+npd+2p$
MMR $^{\diamond}$	DARAM	$n+3$	$n+3$	$n+3+2p$
	SARAM	$n+3$	$n+3$	$n+3+2p$
	External	$2n+2+(n-1)pd$	$2n+2+(n-1)pd$	$2n+4+npd+2p$

# Two operands in same memory block

 $\diamond$  Add  $n$  cycles for peripheral memory-mapped access.

**Class 20B** 3 words, 5 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single program-memory (pmad) write operand.

**Mnemonics** MVDP

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Smem	pmad	ROM/SARAM	DARAM	External
DARAM	DARAM	5	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+2pd+3p
SARAM	DARAM	5, 6†	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+2pd+3p
DROM	DARAM	5, 6†	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+2pd+3p
External	DARAM	5+d	5+d	5+d+3p
	SARAM	5+d	5+d	5+d+3p
	External	5+d+2pd	5+d+2pd	7+d+2pd+3p
MMR <sup>◇</sup>	DARAM	5	5	5+3p
	SARAM	5	5	5+3p
	External	5	5	7+3pd+3p

† Operand and code in same memory block

◇ Add one cycle for peripheral memory-mapped access.

**Class 21A** 2 words, 3 cycles. Single program-memory (pmad) read operand and single data-memory (Smem) write operand.

**Mnemonics** MVPD

**Cycles**

Cycles for a Single Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	3	3	3+2p
	SARAM	3	3	3+2p
	External	3	3	6+d+2p
	MMR <sup>◇</sup>	3	3	3+2p
SARAM	DARAM	3	3	3+2p
	SARAM	3	3	3+2p
	External	3	3	6+d+2p
	MMR <sup>◇</sup>	3	3	3+2p
PROM	DARAM	3	3	3+2p
	SARAM	3	3	3+2p
	External	3	3	6+d+2p
	MMR <sup>◇</sup>	3	3	3+2p
External	DARAM	3+pd	3+pd	3+pd+2p
	SARAM	3+pd	3+pd	3+pd+2p
	External	3+pd	3+pd	6+d+pd+2p
	MMR <sup>◇</sup>	3+pd	3+pd	3+pd+2p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2	n+2	n+2+2p
	External	2n+1+(n-1)d	2n+1+(n-1)d	2n+4+nd+2p
	MMR <sup>◇</sup>	n+2	n+2	n+2+2p
SARAM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2, 2n+1 <sup>#</sup>	n+2, 2n+1 <sup>#</sup>	n+2+2p
	External	2n+1+(n-1)d	2n+1+(n-1)d	2n+4+nd+2p
	MMR <sup>◇</sup>	n+2	n+2	n+2+2p
PROM	DARAM	n+2	n+2	n+2+2p
	SARAM	n+2	n+2	n+2+2p
	External	2n+1+(n-1)d	2n+1+(n-1)d	2n+4+nd+2p
	MMR <sup>◇</sup>	n+2	n+2	n+2+2p
External	DARAM	n+2+npd	n+2+npd	n+2+npd+2p
	SARAM	n+2+npd	n+2+npd	n+2+npd+2p
	External	4n-1+(n-1)d +npd	4n-1+(n-1)d +npd	4n+2+nd+npd+2p
	MMR <sup>◇</sup>	n+2+npd	n+2+npd	n+2+npd+2p

# Two operands in same memory block

◇ Add n cycles for peripheral memory-mapped access.



**Class 21B** 3 words, 4 cycles. Single program-memory (pmad) read operand and single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics** MVPD

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	4	4	4+3p
	SARAM	4	4	4+3p
	External	4	4	7+d+3p
	MMR <sup>◇</sup>	4	4	4+3p
SARAM	DARAM	4	4	4+3p
	SARAM	4	4	4+3p
	External	4	4	7+d+3p
	MMR <sup>◇</sup>	4	4	4+3p
PROM	DARAM	4	4	4+3p
	SARAM	4	4	4+3p
	External	4	4	7+d+3p
	MMR <sup>◇</sup>	4	4	4+3p
External	DARAM	4+2pd	4+2pd	4+2pd+3p
	SARAM	4+2pd	4+2pd	4+2pd+3p
	External	4+2pd	4+2pd	7+d+2pd+3p
	MMR <sup>◇</sup>	4+2pd	4+2pd	4+2pd+3p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 22A** 2 words, 3 cycles. Single data-memory (Smem) read operand and single program-memory (pmad) read operand.

**Mnemonics** MACP

**Cycles**

Cycles for a Single Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	3	3, 4 <sup>†</sup>	3+2p
	SARAM	3, 4 <sup>†</sup>	3	3+2p
	External	3+d	3+d	4+d+2p
	MMR <sup>◇</sup>	3	3	3+2p
SARAM	DARAM	3	3, 4 <sup>†</sup>	3+2p
	SARAM	3, 4 <sup>†</sup>	3	3+2p
	External	3+d	3+d	4+d+2p
	MMR <sup>◇</sup>	3	3	3+2p
PROM	DARAM	3	3, 4 <sup>†</sup>	3+2p
	SARAM	3, 4 <sup>†</sup>	3	3+2p
	External	3+d	3+d	4+d+2p
	MMR <sup>◇</sup>	3	3	3+2p
External	DARAM	3+pd	3+pd, 4+pd <sup>†</sup>	3+pd+2p
	SARAM	3+pd	3+pd	4+pd+2p
	External	4+d+pd	4+d+pd	4+d+pd+2p
	MMR <sup>◇</sup>	3+pd	3+pd	3+pd+2p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	n+2	n+2, n+3†	n+2+2p
	SARAM	n+2, n+3†	n+2	n+2+2p
	External	n+2+nd	n+2+nd	n+2+nd+2p
	MMR <sup>◇</sup>	n+2	n+2	n+2+2p
SARAM	DARAM	n+2	n+2, n+3†	n+2+2p
	SARAM	n+2, n+3†, 2n+2 <sup>#</sup>	n+2, 2n+2 <sup>#</sup>	n+2+2p, 2n+2+2p <sup>#</sup>
	External	n+2+nd	n+2+nd	n+2+nd+2p
	MMR <sup>◇</sup>	n+2	n+2	n+2+2p
PROM	DARAM	n+2	n+2, n+3†	n+2+2p
	SARAM	n+2, n+3†	n+2	n+2+2p
	External	n+2+nd	n+2+nd	n+2+nd+2p
	MMR <sup>◇</sup>	n+2	n+2	n+2+2p
External	DARAM	n+2+npd	n+2+npd, n+3+npd†	n+2+npd+2p
	SARAM	n+2+npd	n+2+npd	n+3+npd+2p
	External	2n+2+nd+npd	2n+2+nd+npd	2n+2+nd+npd+2p
	MMR <sup>◇</sup>	n+2+npd	n+2+npd	n+2+npd+2p

† Operand and code in same memory block

# Two operands in same memory block

◇ Add n cycles for peripheral memory-mapped access.

**Class 22B** 3 words, 4 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single program-memory (pmad) read operand.

**Mnemonics** MACP

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	4	4, 5 <sup>†</sup>	4+3p
	SARAM	4, 5 <sup>†</sup>	4	4+3p
	External	4+d	4+d	5+d+3p
	MMR <sup>◇</sup>	4	4	4+3p
SARAM	DARAM	4	4, 5 <sup>†</sup>	4+3p
	SARAM	4, 5 <sup>†</sup>	4	4+3p
	External	4+d	4+d	5+d+3p
	MMR <sup>◇</sup>	4	4	4+3p
PROM	DARAM	4	4, 5 <sup>†</sup>	4+3p
	SARAM	4, 5 <sup>†</sup>	4	4+3p
	External	4+d	4+d	5+d+3p
	MMR <sup>◇</sup>	4	4	4+3p
External	DARAM	4+2pd	4+2pd, 5+2pd <sup>†</sup>	4+2pd+3p
	SARAM	4+2pd	4+2pd	5+2pd+3p
	External	5+d+2pd	5+d+2pd	5+d+2pd+3p
	MMR <sup>◇</sup>	4+2pd	4+2pd	4+2pd+3p

<sup>†</sup> Operand and code in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 23A** 2 words, 3 cycles. Single data-memory (Smem) read operand, single data-memory (Smem) write operand, and single program-memory (pmad) read operand.

**Mnemonics** MACD

**Cycles**

Cycles for a Single Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	3, 4 <sup>#</sup>	3, 4 <sup>#</sup>	3+2p, 4+2p <sup>#</sup>
	SARAM	3, 4 <sup>†</sup>	3, 4 <sup>†</sup>	3+2p
	External	3+d	3+d	6+2d+2p
	MMR <sup>◇</sup>	3	3	3+2p
SARAM	DARAM	3, 4 <sup>†</sup>	3	3+2p
	SARAM	3, 4 <sup>#</sup>	3, 4 <sup>#</sup>	3+2p, 4+2p <sup>#</sup>
	External	3+d	3+d	6+2d+2p
	MMR <sup>◇</sup>	3	3	3+2p
PROM	DARAM	3	3	3+2p
	SARAM	3, 4 <sup>†</sup>	3	3+2p
	External	3+d	3+d	6+2d+2p
	MMR <sup>◇</sup>	3	3	3+2p
External	DARAM	3+pd	3+pd	3+pd+2p
	SARAM	3+pd	3+pd	3+pd+2p
	External	4+d+pd	4+d+pd	7+d+pd+2p
	MMR <sup>◇</sup>	3+pd	3+pd	4+pd+2p

<sup>†</sup> Operand and code in same memory block

<sup>#</sup> Two operands in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	$n+2, 2n+2^{\#}$	$n+2, 2n+2^{\#}$	$n+2+2p, 2n+2+2p^{\#}$
	SARAM	$n+2, n+3^{\dagger}$	$n+2, n+3^{\dagger}$	$n+2+2p$
	External	$4n+1+2nd$	$4n+1+2nd$	$4n+2+2nd+2p$
	MMR $^{\diamond}$	$n+2$	$n+2$	$n+2+2p$
SARAM	DARAM	$n+2, n+3^{\dagger}$	$n+2$	$n+2+2p$
	SARAM	$n+2, 2n+2^{\#}$	$n+2, 2n+2^{\#}$	$n+2+2p, 2n+2+2p^{\#}$
	External	$4n+1+2nd$	$4n+1+2nd$	$4n+2+2nd+2p$
	MMR $^{\diamond}$	$n+2$	$n+2$	$n+2+2p$
PROM	DARAM	$n+2$	$n+2$	$n+2+2p$
	SARAM	$n+2, n+3^{\dagger}$	$n+2$	$n+2+2p$
	External	$4n+1+2nd$	$4n+1+2nd$	$4n+2+2nd+2p$
	MMR $^{\diamond}$	$n+2$	$n+2$	$n+2+2p$
External	DARAM	$n+2+npd$	$n+2+npd, n+3+npd^{\dagger}$	$n+2+npd+2p$
	SARAM	$n+2+npd$	$n+2+npd$	$n+2+npd+2p$
	External	$5n-1+nd+npd$	$5n-1+nd+npd$	$5n+2+nd+npd+2p$
	MMR $^{\diamond}$	$n+2+npd$	$n+2+npd$	$4n+3+npd+2p$

† Operand and code in same memory block

# Two operands in same memory block

◊ Add one cycle for peripheral memory-mapped access.

**Class 23B** 3 words, 4 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing, single data-memory (Smem) write operand using long-offset indirect addressing, and single program-memory (pmad) read operand.

**Mnemonics** MACD

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	4, 5 <sup>#</sup>	4, 5 <sup>#</sup>	4+3p, 5+3p <sup>#</sup>
	SARAM	4, 5 <sup>†</sup>	4, 5 <sup>†</sup>	4+3p
	External	4+d	4+d	7+2d+3p
	MMR <sup>◇</sup>	4	4	4+3p
SARAM	DARAM	4, 5 <sup>†</sup>	4	4+3p
	SARAM	4, 5 <sup>#</sup>	4, 5 <sup>#</sup>	4+3p, 5+3p <sup>#</sup>
	External	4+d	4+d	7+2d+3p
	MMR <sup>◇</sup>	4	4	4+3p
PROM	DARAM	4	4	4+3p
	SARAM	4, 5 <sup>†</sup>	4	4+3p
	External	4+d	4+d	7+2d+3p
	MMR <sup>◇</sup>	4	4	4+3p
External	DARAM	4+2pd	4+2pd	4+pd+3p
	SARAM	4+2pd	4+2pd	4+2pd+3p
	External	5+d+2pd	5+d+2pd	8+d+2pd+3p
	MMR <sup>◇</sup>	4+2pd	4+2pd	5+2pd+3p

<sup>†</sup> Operand and code in same memory block

<sup>#</sup> Two operands in same memory block

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 24A** 1 word, 1 cycle. Single data-memory (Smem) read operand and single data-memory (Smem) write operand.

**Mnemonics** DELAY LTD

**Cycles**

Cycles for a Single Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	1	1, 2†	1+p
SARAM	1, 3†	1	1+p
External	1+d	1+d	5+p+2d

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	n	n, n+1†	n+p
SARAM	2n-1, 2n+1†	2n-1	2n-1+p
External	4n-3+(2n-1)d	4n-3+(2n-1)d	4n+1+p+2nd

† Operand and code in same memory block

**Class 24B** 2 words, 2 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics** DELAY LTD

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier

Operand	Program		
	ROM/SARAM	DARAM	External
Smem			
DARAM	2	2, 3†	2+2p
SARAM	2, 4†	2	2+2p
External	2+d	2+d	6+2p+2d

† Operand and code in same memory block



**Class 25A** 1 word, 5 cycles. Single program-memory (pmad) read address and single data-memory (Smem) write operand.

**Mnemonics** READA

**Cycles**

Cycles for a Single Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	8+d+p
	MMR <sup>◇</sup>	5	5	5+p
SARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	8+d+p
	MMR <sup>◇</sup>	5	5	5+p
PROM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	8+d+p
	MMR <sup>◇</sup>	5	5	5+p
External	DARAM	5+pd	5+pd	5+pd+p
	SARAM	5+pd	5+pd	5+pd+p
	External	5+pd	5+pd	8+pd+d+p
	MMR <sup>◇</sup>	5+pd	5+pd	5+pd+p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	$2n+3+(n-1)d$	$2n+3+(n-1)d$	$2n+6+nd+np$
	MMR <sup>◇</sup>	n+4	n+4	n+4+p
SARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4, $2n+3^{\#}$	n+4, $2n+3^{\#}$	n+4+p, $2n+3+p^{\#}$
	External	$2n+3+(n-1)d$	$2n+3+(n-1)d$	$2n+6+nd+p$
	MMR <sup>◇</sup>	n+4	n+4	n+4+p
PROM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	$2n+3+(n-1)d$	$2n+3+(n-1)d$	$2n+6+nd+p$
	MMR <sup>◇</sup>	n+4	n+4	n+4+p
External	DARAM	n+4+npd	n+4+npd	n+4+npd+p
	SARAM	n+4+npd	n+4+npd	n+4+npd+p
	External	$4n+1+(n-1)d$ +npd	$4n+1+(n-1)d$ +npd	$4n+4+nd+npd$ +p
	MMR <sup>◇</sup>	n+4+npd	n+4+npd	n+4+npd+p

<sup>#</sup> Two operands in same memory block

<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

**Class 25B** 2 words, 6 cycles. Single program-memory (pmad) read address and single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics** READA

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
pmad	Smem	ROM/SARAM	DARAM	External
DARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	9+d+2p
	MMR <sup>◇</sup>	6	6	6+2p
SARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	9+d+2p
	MMR <sup>◇</sup>	6	6	6+2p
PROM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	9+d+2p
	MMR <sup>◇</sup>	6	6	6+2p
External	DARAM	6+2pd	6+2pd	6+2pd+2p
	SARAM	6+2pd	6+2pd	6+2pd+2p
	External	6+2pd	6+2pd	9+2pd+d+2p
	MMR <sup>◇</sup>	6+2pd	6+2pd	6+2pd+2p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 26A** 1 word, 5 cycles. Single data-memory (Smem) read operand and single program-memory (pmad) write address.

**Mnemonics** WRITA

**Cycles**

Cycles for a Single Execution				
Operand		Program		
Smem	pmad	ROM/SARAM	DARAM	External
DARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p
SARAM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p
DROM	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p
External	DARAM	5+pd	5+pd	5+pd+p
	SARAM	5+pd	5+pd	5+pd+p
	External	5+d	5+d	7+d+pd+p
MMR <sup>◇</sup>	DARAM	5	5	5+p
	SARAM	5	5	5+p
	External	5	5	5+pd+p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

Cycles for a Repeat Execution

Operand		Program		
Smem	pmad	ROM/SARAM	DARAM	External
DARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n-1)pd	2n+3+(n-1)pd	2n+3+npd+p
SARAM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4, 2n+3 <sup>#</sup>	n+4, 2n+3 <sup>#</sup>	n+4+p, 2n+3+p <sup>#</sup>
	External	2n+3+(n-1)pd	2n+3+(n-1)pd	2n+3+npd+p
DROM	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n-1)pd	2n+3+(n-1)pd	2n+3+npd+p
External	DARAM	n+4+npd	n+4+npd	n+4+npd+p
	SARAM	n+4+npd	n+4+npd	n+4+npd+p
	External	4n+1+nd +(n-1)pd	4n+1+nd +(n-1)pd	4n+3+nd+npd +p
MMR <sup>◇</sup>	DARAM	n+4	n+4	n+4+p
	SARAM	n+4	n+4	n+4+p
	External	2n+3+(n-1)pd	2n+3+(n-1)pd	2n+3+npd+p

<sup>#</sup> Two operands in same memory block<sup>◇</sup> Add n cycles for peripheral memory-mapped access.

## Class 26B

**Class 26B** 2 words, 6 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single program-memory (pmad) write address.

**Mnemonics** WRITA

### Cycles

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
Smem	pmad	ROM/SARAM	DARAM	External
DARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p
SARAM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p
DROM	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p
External	DARAM	6+2pd	6+2pd	6+2pd+2p
	SARAM	6+2pd	6+2pd	6+2pd+2p
	External	6+d	6+d	8+d+2pd+2p
MMR <sup>◇</sup>	DARAM	6	6	6+2p
	SARAM	6	6	6+2p
	External	6	6	6+2pd+2p

<sup>◇</sup> Add one cycle for peripheral memory-mapped access.

**Class 27A** 2 words, 2 cycles. Single I/O port read operand and single data-memory (Smem) write operand.

**Mnemonics** PORTR

**Cycles**

Cycles for a Single Execution

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	3+io	3+io	6+2p+io
	SARAM	3+io, 4+io <sup>†</sup>	3+io	6+2p+io
	External	3+io	3+io	9+2p+d+io

<sup>†</sup> Operand and code in same memory block

Cycles for a Repeat Execution

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	2n+1+nio	2n+1+nio	2n+4+2p+nio
	SARAM	2n+1+nio, 2n+2+nio <sup>†</sup>	2n+1+nio	2n+4+2p+nio
	External	5n-2+nio +(n-1)d	5n-2+nio +(n-1)d	5n+4+2p +nio+nd

<sup>†</sup> Operand and code in same memory block

**Class 27B** 3 words, 3 cycles. Single I/O port read operand and single data-memory (Smem) write operand using long-offset indirect addressing.

**Mnemonics** PORTR

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	4+io	4+io	7+3p+io
	SARAM	4+io, 5+io <sup>†</sup>	4+io	7+3p+io
	External	4+io	4+io	10+3p+d+io

<sup>†</sup> Operand and code in same memory block

**Class 28A** 2 words, 2 cycles. Single data-memory (Smem) read operand and single I/O port write operand.

**Mnemonics** PORTW

**Cycles**

Cycles for a Single Execution

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	2	2, 3†	6+2p+io
	SARAM	2, 3†	2	6+2p+io
	DROM	2, 3†	2	6+2p+io
	External	2+d	2+d	7+2p+d+io

† Operand and code in same memory block

Cycles for a Repeat Execution

Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	$2n+(n-1)io$	$2n+(n-1)io, 2n+1+(n-1)io†$	$2n+4+2p+nio$
	SARAM	$2n+(n-1)io, 2n+1+(n-1)io†$	$2n+(n-1)io$	$2n+4+2p+nio$
	DROM	$2n+(n-1)io, 2n+1+(n-1)io†$	$2n+(n-1)io$	$2n+4+2p+nio$
	External	$5n-3+nd+(n-1)io$	$5n-3+nd+(n-1)io$	$5n+2+2p+nd+nio$

† Operand and code in same memory block



**Class 28B** 3 words, 3 cycles. Single data-memory (Smem) read operand using long-offset indirect addressing and single I/O port write operand.

**Mnemonics** PORTW

**Cycles**

Cycles for a Single Execution With Long-Offset Modifier				
Operand		Program		
Port	Smem	ROM/SARAM	DARAM	External
External	DARAM	3	3, 4†	7+3p+io
	SARAM	3, 4†	3	7+3p+io
	DROM	3, 4†	3	7+3p+io
	External	3+d	3+d	8+3p+d+io

† Operand and code in same memory block

**Class 29A** 2 words, 4 cycles, 2 cycles (delayed), 2 cycles (false condition). Single program-memory (pmad) operand.

**Mnemonics**

B[D]                      BANZ[D]                      FB[D]                      RPTB[D]

**Cycles**

Cycles for a Single Execution		
Program		
ROM/SARAM	DARAM	External
4	4	4+4p

Cycles for a Single Delayed Execution		
Program		
ROM/SARAM	DARAM	External
2	2	2+2p

**Class 29B** 2 words, 4 cycles, 2 cycles (delayed). Single program-memory (pmad) operand.

**Mnemonics**

CALL[D]                      FCALL[D]

**Cycles**

Cycles for a Single Execution			
Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	4	4	4+4p
SARAM	4, 5†	4	4+4p
External	4	4	7+4p+d

† Operand and code in same memory block

Cycles for a Single Delayed Execution			
Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	2	2	2+2p
SARAM	2, 3†	2	2+2p
External	2	2	5+2p+d

† Operand and code in same memory block

**Class 30A** 1 word, 6 cycles, 4 cycles (delayed). Single register operand.

**Mnemonics** BACC[D] FBACC[D]

**Cycles**

Cycles for a Single Execution		
Program		
ROM/SARAM	DARAM	External
6	6	6+3p

Cycles for a Single Delayed Execution		
Program		
ROM/SARAM	DARAM	External
4	4	4+p

**Class 30B** 1 word, 6 cycles, 4 cycles (delayed). Single register operand.

**Mnemonics** CALA[D] FCALA[D]

**Cycles**

Cycles for a Single Execution			
Stack	Program		
	ROM/SARAM	DARAM	External
DARAM	6	6	6+3p
SARAM	6	6	6+3p
External	6	6	7+3p+d

Cycles for a Single Delayed Execution			
Stack	Program		
	ROM/SARAM	DARAM	External
DARAM	4	4	4+p
SARAM	4	4	4+p
External	4	4	5+p+d

### Class 31A

---

**Class 31A** 2 words, 5 cycles, 3 cycles (delayed). Single program-memory (pmad) operand and short-immediate operands.

**Mnemonics** BC[D]

**Cycles**

Cycles for a Single Execution			
Condition	Program		
	ROM/SARAM	DARAM	External
True	5	5	5+4p
False	3	3	3+2p

Cycles for a Single Delayed Execution			
Condition	Program		
	ROM/SARAM	DARAM	External
True	3	3	3+2p
False	3	3	3+2p

**Class 31B** 2 words, 5 cycles, 3 cycles (delayed), 3 cycles (false condition). Single program-memory (pmad) operand and short-immediate operands.

**Mnemonics** CC[D]

**Cycles**

Cycles for a Single True Condition Execution

Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	5	5	5+4p
SARAM	5, 6†	5	5+4p
External	5	5	8+4p+d

† Operand and code in same memory block

Cycles for a Single False Condition Execution

Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	3	3	3+2p
SARAM	3, 4†	3	3+2p
External	3	3	6+2p+d

† Operand and code in same memory block

Cycles for a Single Delayed Execution

Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	3	3	3+2p
SARAM	3, 4†	3	3+2p
External	3	3	6+2p+d

† Operand and code in same memory block

**Class 32**

**Class 32** 1 word, 5 cycles, 3 cycles (delayed), 3 cycles (false condition). No operand, or short-immediate operands.

**Mnemonics** RC[D] RET[D] RETE[D]

**Cycles**

**Cycle Timings for a Single Execution**

Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	5	5, 6†	5+3p
SARAM	5, 6†	5	5+3p
External	5+d	5+d	6+d+3p

† Operand and code in same memory block

**Cycles for a Single Delayed Execution**

Operand	Program		
Stack	ROM/SARAM	DARAM	External
DARAM	3	3, 4†	3+p
SARAM	3, 4†	3	3+p
External	3+d	3+d	4+d+p

† Operand and code in same memory block

**Class 33** 1 word, 3 cycles, 1 cycle (delayed). No operand.

**Mnemonics** RETF[D]

**Cycles**

Cycles for a Single Execution		
Program		
ROM/SARAM	DARAM	External
3	3	3+p

Cycles for a Single Delayed Execution		
Program		
ROM/SARAM	DARAM	External
1	1	1+p

**Class 34** 1 word, 6 cycles, 4 cycles (delayed). No operand.

**Mnemonics** FRET[D] FRETE[D]

**Cycles**

Cycles for a Single Execution			
Stack	Program		
	ROM/SARAM	DARAM	External
DARAM	6	6, 8†	6+3p
SARAM	6, 8†	6	6+3p
External	6+2d	6+2d	8+3p+d

† Operand and code in same memory block

Cycles for a Single Delayed Execution			
Stack	Program		
	ROM/SARAM	DARAM	External
DARAM	4	4, 6†	4+p
SARAM	4, 6†	4	4+p
External	4+2d	4+2d	6+p+2d

† Operand and code in same memory block

**Class 35** 1 word, 3 cycles. No operand or single short-immediate operand.

**Mnemonics** INTR RESET TRAP

**Cycles**

Cycles for a Single Execution		
Program		
ROM/SARAM	DARAM	External
3	3	3+p

**Class 36** 1 word, 4 cycles (minimum). Single short-immediate operand.

**Mnemonics** IDLE

**Cycles** The number of cycles needed to execute this instruction depends on the idle period.



# Assembly Language Instructions

This section provides detailed information on the instruction set for the '54x family. The '54x instruction set supports numerically intensive signal-processing operations as well as general-purpose applications, such as multiprocessing and high-speed control.

See Section 1.1, *Instruction Set Symbols and Abbreviations*, for definitions of symbols and abbreviations used in the description of assembly language instructions. See Section 1.2, *Example Description of Instruction*, for a description of the elements in an instruction. See Chapter 2 for a summary of the instruction set.

## ABDST *Absolute Distance*

**Syntax** ABDST *Xmem, Ymem*

**Operands** Xmem, Ymem: Dual data-memory operands

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	1	X	X	X	X	Y	Y	Y	Y

**Execution**  $(B) + |(A(32-16))| \rightarrow B$   
 $((Xmem) - (Ymem)) \ll 16 \rightarrow A$

**Status Bits** Affected by OVM, FRCT, and SXM  
 Affects C, OVA, and OVB

**Description** This instruction calculates the absolute value of the distance between two vectors, *Xmem* and *Ymem*. The absolute value of accumulator A(32-16) is added to accumulator B. The content of *Ymem* is subtracted from *Xmem*, and the result is left-shifted 16 bits and stored in accumulator A. If the fractional mode bit is logical 1 (FRCT = 1), the absolute value is multiplied by 2.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 7 (see page 3-12)

**Example** ABDST \*AR3+, \*AR4+

	Before Instruction	After Instruction
A	FF ABCD 0000	FF FFAB 0000
B	00 0000 0000	00 0000 5433
AR3	0100	0101
AR4	0200	0201
FRCT	0	0
Data Memory		
0100h	0055	0055
0200h	00AA	00AA

**Syntax** **ABS** *src* [, *dst*]

**Operands** *src, dst:* A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	0	0	1	0	1

**Execution** |(src)| → dst (or src if dst is not specified)

**Status Bits** OVM affects this instruction as follows:

If OVM = 1, the absolute value of 80 0000 0000h is 00 7FFF FFFFh.

If OVM = 0, the absolute value of 80 0000 0000h is 80 0000 0000h.

Affects C and OVdst (or OVsrc, if dst = src)

**Description** This instruction calculates the absolute value of *src* and loads the value into *dst*. If no *dst* is specified, the absolute value is loaded into *src*.

If the result of the operation is equal to 0, the carry bit, C, is set.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1** ABS A, B

Before Instruction			After Instruction		
A	FF FFFF FFCB	-53	A	FF FFFF FFCB	-53
B	FF FFFF FC18	-1000	B	00 0000 0035	+53

**Example 2** ABS A

Before Instruction			After Instruction		
A	03 1234 5678		A	00 7FFF FFFF	
OVM		1	OVM		1

**Example 3** ABS A

Before Instruction			After Instruction		
A	03 1234 5678		A	03 1234 5678	
OVM		0	OVM		0

## ADD Add to Accumulator

### Syntax

- 1: **ADD** *Smem*, *src*
- 2: **ADD** *Smem*, **TS**, *src*
- 3: **ADD** *Smem*, **16**, *src* [, *dst*]
- 4: **ADD** *Smem* [, *SHIFT*], *src* [, *dst*]
- 5: **ADD** *Xmem*, *SHFT*, *src*
- 6: **ADD** *Xmem*, *Ymem*, *dst*
- 7: **ADD** *#lk* [, *SHFT*], *src* [, *dst*]
- 8: **ADD** *#lk*, **16**, *src* [, *dst*]
- 9: **ADD** *src* [, *SHIFT*], [, *dst*]
- 10: **ADD** *src*, **ASM** [, *dst*]

### Operands

*Smem*: Single data-memory operand  
*Xmem*, *Ymem*: Dual data-memory operands  
*src*, *dst*: A (accumulator A)  
               B (accumulator B)

$-32\,768 \leq lk \leq 32\,767$

$-16 \leq SHIFT \leq 15$

$0 \leq SHFT \leq 15$

### Opcode

- 1:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	S	I	A	A	A	A	A	A	A
- 2:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	S	I	A	A	A	A	A	A	A
- 3:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	S	D	I	A	A	A	A	A	A	A
- 4:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	S	D	0	0	0	S	H	I	F	T
- 5:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	S	X	X	X	X	S	H	F	T
- 6:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	D	X	X	X	X	Y	Y	Y	Y
- 7:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	0	0	0	S	H	F	T
16-bit constant															

## Add to Accumulator **ADD**

8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	0	0	0
16-bit constant															

9:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	0	0	S	H	I	F	T

10:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	0	0	0	0	0

### Execution

- 1: (Smem) + (src) → src
- 2: (Smem) << (TS) + (src) → src
- 3: (Smem) << 16 + (src) → dst
- 4: (Smem) [<< SHIFT] + (src) → dst
- 5: (Xmem) << SHFT + (src) → src
- 6: ((Xmem) + (Ymem)) << 16 → dst
- 7: lk << SHFT + (src) → dst
- 8: lk << 16 + (src) → dst
- 9: (src or [dst]) + (src) << SHIFT → dst
- 10: (src or [dst]) + (src) << ASM → dst

### Status Bits

Affected by SXM and OVM  
Affects C and OVdst (or OVsrc, if dst = src)

For instruction syntax 3, if the result of the addition generates a carry, the carry bit, C, is set to 1; otherwise, C is not affected.

## **ADD** *Add to Accumulator*

---

### **Description**

This instruction adds a 16-bit value to the content of the selected accumulator or to a 16-bit operand *Xmem* in dual data-memory operand addressing mode. The 16-bit value added is one of the following:

- ☐ The content of a single data-memory operand (*Smem*)
- ☐ The content of a dual data-memory operand (*Ymem*)
- ☐ A 16-bit immediate operand (*#k*)
- ☐ The shifted value in *src*

If *dst* is specified, this instruction stores the result in *dst*. If no *dst* is specified, this instruction stores the result in *src*. Most of the second operands can be shifted. For a left shift:

- ☐ Low-order bits are cleared
- ☐ High-order bits are:
  - Sign extended if *SXM* = 1
  - Cleared if *SXM* = 0

For a right shift, the high-order bits are:

- Sign extended if *SXM* = 1
- Cleared if *SXM* = 0

---

### **Notes:**

The following syntaxes are assembled as a different syntax in certain cases.

- ☐ Syntax 4: If *dst* = *src* and *SHIFT* = 0, then the instruction opcode is assembled as syntax 1.
  - ☐ Syntax 4: If *dst* = *src*, *SHIFT* ≤ 15 and *Smem* indirect addressing mode is included in *Xmem*, then the instruction opcode is assembled as syntax 5.
  - ☐ Syntax 5: If *SHIFT* = 0, the instruction opcode is assembled as syntax 1.
- 

### **Words**

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 word  
Syntaxes 4, 7, and 8: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

### **Cycles**

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 cycle  
Syntaxes 4, 7, and 8: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

## Classes

Syntaxes 1, 2, 3, and 5: Class 3A (see page 3-5)  
 Syntaxes 1, 2, and 3: Class 3B (see page 3-6)  
 Syntax 4: Class 4A (see page 3-7)  
 Syntax 4: Class 4B (see page 3-8)  
 Syntax 6: Class 7 (see page 3-12)  
 Syntaxes 7 and 8: Class 2 (see page 3-4)  
 Syntaxes 9 and 10: Class 1 (see page 3-3)

## Example 1

ADD \*AR3+, 14, A

	Before Instruction	After Instruction
A	00 0000 1200	00 0540 1200
C	1	0
AR3	0100	0101
SXM	1	1
Data Memory		
0100h	1500	1500

## Example 2

ADD A, -8, B

	Before Instruction	After Instruction
A	00 0000 1200	00 0000 1200
B	00 0000 1800	00 0000 1812
C	1	0

## Example 3

ADD #4568, 8, A, B

	Before Instruction	After Instruction
A	00 0000 1200	00 0000 1200
B	00 0000 1800	00 0011 EA00
C	1	0

## ADDC *Add to Accumulator With Carry*

**Syntax** **ADDC** *Smem, src*

**Operands** **Smem:** Single data-memory operand  
**src:** A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	S	I	A	A	A	A	A	A	A

**Execution** (Smem) + (src) + (C) → src

**Status Bits** Affected by OVM, C  
Affects C and OVsrc

**Description** This instruction adds the 16-bit single data-memory operand *Smem* and the value of the carry bit (C) to *src*. This instruction stores the result in *src*. Sign extension is suppressed regardless of the value of the SXM bit.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 3A (see page 3-5)  
Class 3B (see page 3-6)

**Example** **ADDC** *\*+AR2(5), A*

	Before Instruction	After Instruction
A	00 0000 0013	00 0000 0018
C	1	0
AR2	0100	0105
Data Memory		
0105h	0004	0004



**Syntax** **ADDM** *#lk, Smem*

**Operands** Smem: Single data-memory operand  
 $-32\,768 \leq lk \leq 32\,767$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	1	I	A	A	A	A	A	A	A
16-bit constant															

**Execution**  $\#lk + (Smem) \rightarrow Smem$

**Status Bits** Affected by OVM and SXM  
 Affects C and OVA

**Description** This instruction adds the 16-bit single data-memory operand *Smem* to the 16-bit immediate memory value *lk* and stores the result in *Smem*.

**Note:**

This instruction is not repeatable.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 18A (see page 3-39)  
 Class 18B (see page 3-39)

**Example 1** ADDM 0123Bh, \*AR4+

	Before Instruction	After Instruction
AR4	0100	0101
Data Memory		
0100h	0004	123F

**Example 2** ADDM 0FFF8h, \*AR4+

	Before Instruction	After Instruction
OVM	1	1
SXM	1	1
AR4	0100	0101
Data Memory		
0100h	8007	8000

## **ADDS** *Add to Accumulator With Sign-Extension Suppressed*

**Syntax** **ADDS** *Smem*, *src*

**Operands** *Smem*: Single data-memory operands  
*src*: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	S	I	A	A	A	A	A	A	A

**Execution**  $\text{uns}(\text{Smem}) + (\text{src}) \rightarrow \text{src}$

**Status Bits** Affected by OVM  
Affects C and OVsrc

**Description** This instruction adds the 16-bit single data-memory operand *Smem* to *src* and stores the result in *src*. Sign extension is suppressed regardless of the value of the SXM bit.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 3A (see page 3-5)  
Class 3B (see page 3-6)

**Example** **ADDS** \*AR2-, B

	Before Instruction	After Instruction
B	00 0000 0003	00 0000 F009
C	x	0
AR2	0100	00FF
Data Memory		
0104h	F006	F006

Syntax	1: AND Smem, src 2: AND #lk [, SHFT], src [, dst] 3: AND #lk, 16, src [, dst] 4: AND src [, SHIFT], [, dst]																																																																																																																																																																
Operands	Smem: Single data-memory operand src: A (accumulator A) B (accumulator B) -16 ≤ SHIFT ≤ 15 0 ≤ SHFT ≤ 15 0 ≤ lk ≤ 65 535																																																																																																																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>D</td><td>0</td><td>0</td><td>1</td><td>1</td><td>S</td><td>H</td><td>F</td><td>T</td></tr><tr><td colspan="16">16-bit constant</td></tr></table> 3: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>D</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td colspan="16">16-bit constant</td></tr></table> 4: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>D</td><td>1</td><td>0</td><td>0</td><td>S</td><td>H</td><td>I</td><td>F</td><td>T</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	1	1	0	0	S	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	S	D	0	0	1	1	S	H	F	T	16-bit constant																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	S	D	0	1	1	0	0	0	1	1	16-bit constant																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	S	D	1	0	0	S	H	I	F	T
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
0	0	0	1	1	0	0	S	I	A	A	A	A	A	A	A																																																																																																																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
1	1	1	1	0	0	S	D	0	0	1	1	S	H	F	T																																																																																																																																																		
16-bit constant																																																																																																																																																																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
1	1	1	1	0	0	S	D	0	1	1	0	0	0	1	1																																																																																																																																																		
16-bit constant																																																																																																																																																																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
1	1	1	1	0	0	S	D	1	0	0	S	H	I	F	T																																																																																																																																																		
Execution	1: (Smem) AND (src) → src 2: lk << SHFT AND (src)→ dst 3: lk << 16 AND (src)→ dst 4: (dst) AND (src) << SHIFT → dst																																																																																																																																																																
Status Bits	None																																																																																																																																																																
Description	This instruction ANDs the following to src: <input type="checkbox"/> A 16-bit operand Smem <input type="checkbox"/> A 16-bit immediate operand lk <input type="checkbox"/> The source or destination accumulator (src or dst)																																																																																																																																																																

If a shift is specified, this instruction left-shifts the operand before the AND. For a left shift, the low-order bits are cleared and the high-order bits are not sign extended. For a right shift, the high-order bits are not sign extended.

## AND AND With Accumulator

---

### Words

Syntaxes 1 and 4: 1 word  
Syntaxes 2 and 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

### Cycles

Syntaxes 1 and 4: 1 cycle  
Syntaxes 2 and 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

### Classes

Syntax 1: Class 3A (see page 3-5)  
Syntax 1: Class 3B (see page 3-6)  
Syntaxes 2 and 3: Class 2 (see page 3-4)  
Syntax 4: Class 1 (see page 3-3)

### Example 1

AND \*AR3+, A

Before Instruction		After Instruction	
A	00 00FF 1200	A	00 0000 1000
AR3	0100	AR3	0101
Data Memory			
0100h	1500	0100h	1500

### Example 2

AND A, 3, B

Before Instruction		After Instruction	
A	00 0000 1200	A	00 0000 1200
B	00 0000 1800	B	00 0000 1000

**Syntax** **ANDM #lk, Smem****Operands** Smem: Single data-memory operand  
 $0 \leq lk \leq 65\,535$ 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	0	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** lk AND (Smem) → Smem**Status Bits** None**Description** This instruction ANDs the 16-bit single data-memory operand *Smem* with a 16-bit long constant *lk*. The result is stored in the data-memory location specified by *Smem*.**Note:**

This instruction is not repeatable.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 18A (see page 3-39)  
Class 18B (see page 3-39)**Example 1** ANDM #00FFh, \*AR4+

	Before Instruction	After Instruction
AR4	0100	0101
Data Memory		
0100h	0444	0044

**Example 2** ANDM #0101h, 4

	Before Instruction	After Instruction
Data Memory		
0004h	00 0000 0100	00 0000 0100

## B[D] Branch Unconditionally

**Syntax** B[D] *pmad*

**Operands**  $0 \leq pmad \leq 65535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	Z	0	0	1	1	1	0	0	1	1
16-bit constant															

**Execution** *pmad* → PC

**Status Bits** None

**Description** This instruction passes control to the designated program-memory address (*pmad*), which can be either a symbolic or numeric address. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words** 2 words

**Cycles** 4 cycles  
2 cycles (delayed)

**Classes** Class 29A (see page 3-66)

**Example 1** B 2000h

	Before Instruction		After Instruction
PC	1F45	PC	2000

**Example 2** BD 1000h  
ANDM 4444h, \*AR1+

	Before Instruction		After Instruction
PC	1F45	PC	1000

After the operand has been ANDed with 4444h, the program continues executing from location 1000h.

**Syntax**                    **BACC[D] src**

**Operands**                src:     A (accumulator A)  
     B (accumulator B)

**Opcode**                    15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0  
                                  1 1 1 1 0 1 Z S 1 1 1 0 0 0 1 0

**Execution**                (src(15–0)) → PC

**Status Bits**              None

**Description**             This instruction passes control to the 16-bit address in the low part of *src* (bits 15–0). If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words**                    1 word

**Cycles**                    6 cycles  
                                  4 cycles (delayed)

**Classes**                    Class 30A (see page 3-67)

**Example 1**                **BACC A**

	Before Instruction		After Instruction
A	00 0000 3000		00 0000 3000
PC	1F45		3000

**Example 2**                **BACCD B**  
**ANDM 4444h, \*AR1+**

	Before Instruction		After Instruction
B	00 0000 2000		00 0000 2000
PC	1F45		2000

After the operand has been ANDed with 4444h value, the program continues executing from location 2000h.

## **BANZ[D]** *Branch on Auxiliary Register Not Zero*

**Syntax** **BANZ[D]** *pmad, Sind*

**Operands** Sind: Single indirect addressing operand  
 $0 \leq \text{pmad} \leq 65\,535$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	Z	0	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** If  $((\text{ARx}) \neq 0)$   
Then  
     $\text{pmad} \rightarrow \text{PC}$   
Else  
     $(\text{PC}) + 2 \rightarrow \text{PC}$

**Status Bits** None

**Description** This instruction branches to the specified program-memory address (*pmad*) if the value of the current auxiliary register ARx is not 0. Otherwise, the PC is incremented by 2. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

### **Note:**

This instruction is not repeatable.

**Words** 2 words

**Cycles** 4 cycles (true condition)  
2 cycles (false condition)  
2 cycles (delayed)

**Classes** Class 29A (see page 3-66)

**Example 1** BANZ 2000h, \*AR3-

Before Instruction		After Instruction	
PC	1000	PC	2000
AR3	0005	AR3	0004

**Example 2** BANZ 2000h, \*AR3-

Before Instruction		After Instruction	
PC	1000	PC	1002
AR3	0000	AR3	FFFF



**Example 3**

BANZ 2000h, \*AR3 (-1)

	Before Instruction		After Instruction
PC	1000	PC	1002
AR3	0001	AR3	0001

**Example 4**

BANZD 2000h, \*AR3-

ANDM 4444h, \*AR5+

	Before Instruction		After Instruction
PC	1000	PC	2000
AR3	0004	AR3	0003

After the memory location has been ANDed with 4444h, the program continues executing from location 2000h.

**BC[D]** *Branch Conditionally***Syntax** **BC[D]** *pmad*, *cond* [, *cond* [, *cond* ]]**Operands**  $0 \leq \text{pmad} \leq 65\,535$ The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{\text{BIO}}$ low	0000 0011	NBIO	$\overline{\text{BIO}}$ high	0000 0010
C	$C = 1$	0000 1100	NC	$C = 0$	0000 1000
TC	$\text{TC} = 1$	0011 0100	NTC	$\text{TC} = 0$	0010 0000
AEQ	$(A) = 0$	0100 0101	BEQ	$(B) = 0$	0100 1101
ANEQ	$(A) \neq 0$	0100 0100	BNEQ	$(B) \neq 0$	0100 1100
AGT	$(A) > 0$	0100 0110	BGT	$(B) > 0$	0100 1110
AGEQ	$(A) \geq 0$	0100 0010	BGEQ	$(B) \geq 0$	0100 1010
ALT	$(A) < 0$	0100 0011	BLT	$(B) < 0$	0100 1011
ALEQ	$(A) \leq 0$	0100 0111	BLEQ	$(B) \leq 0$	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	Z	0	C	C	C	C	C	C	C	C
16-bit constant															

**Execution**If (*cond*(s))

Then

 $\text{pmad} \rightarrow \text{PC}$ 

Else

 $(\text{PC}) + 2 \rightarrow \text{PC}$ **Status Bits**

Affects OVA or OVB if OV or NOV is chosen

**Description**

This instruction branches to the program-memory address (*pmad*) if the specified condition(s) is met. The two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed. If the condition(s) is not met, the PC is incremented by 2. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction has no effect on the condition tested.

This instruction tests multiple conditions before passing control to another section of the program. This instruction can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

- Group 1:** You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time.
- Group 2:** You can select up to three conditions. Each of these conditions must be from a different category (category A, B, or C); you cannot have two conditions from the same category. For example, you can test TC, C, and BIO at the same time but you cannot test NTC, C, and NC at the same time.

Conditions for This Instruction				
Group 1		Group 2		
Category A	Category B	Category A	Category B	Category C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

**Note:**

This instruction is not repeatable.

<b>Words</b>	2 words
<b>Cycles</b>	5 cycles (true condition) 3 cycles (false condition) 3 cycles (delayed)
<b>Classes</b>	Class 31A (see page 3-68)
<b>Example 1</b>	BC 2000h, AGT

	Before Instruction		After Instruction
A	00 0000 0053	A	00 0000 0053
PC	1000	PC	2000

## BC[D] Branch Conditionally

### Example 2

BC 2000h, AGT

Before Instruction		After Instruction	
A	FF FFFF FFFF	A	FF FFFF FFFF
PC	1000	PC	1002

### Example 3

BCD 1000h, BOV  
ANDM 4444h, \*AR1+

Before Instruction		After Instruction	
PC	3000	PC	1000
OVB	1	OVB	1

After the memory location is ANDed with 4444h, the branch is taken if the condition (OVB) is met. Otherwise, execution continues at the instruction following this instruction.

### Example 4

BC 1000h, TC, NC, BIO

Before Instruction		After Instruction	
PC	3000	PC	3002
C	1	C	1

Assembly Language Instructions 4-21

## **BITF** *Test Bit Field Specified by Immediate Value*

**Syntax** **BITF** *Smem*, #*lk*

**Operands** *Smem*: Single data-memory operand  
 $0 \leq lk \leq 65\,535$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** If  $((Smem) \text{ AND } lk) = 0$   
Then  
     $0 \rightarrow TC$   
Else  
     $1 \rightarrow TC$

**Status Bits** Affects TC

**Description** This instruction tests the specified bit or bits of the data-memory value *Smem*. If the specified bit (or bits) is 0, the TC bit in status register ST0 is cleared to 0; otherwise, TC is set to 1. The *lk* constant is a mask for the bit or bits tested.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 6A (see page 3-10)  
Class 6B (see page 3-11)

**Example 1** **BITF** DAT5, 00FFh

Before Instruction		After Instruction	
TC	<input type="text" value="x"/>	TC	<input type="text" value="0"/>
DP	<input type="text" value="004"/>	DP	<input type="text" value="004"/>
Data Memory			
0205h	<input type="text" value="5400"/>	0205h	<input type="text" value="5400"/>

**Example 2** **BITF** DAT5, 0800h

Before Instruction		After Instruction	
TC	<input type="text" value="x"/>	TC	<input type="text" value="1"/>
DP	<input type="text" value="004"/>	DP	<input type="text" value="004"/>
Data Memory			
0205h	<input type="text" value="0F7F"/>	0205h	<input type="text" value="0F7F"/>

**Syntax** **BITT** *Smem*

**Operands** *Smem*: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	0	I	A	A	A	A	A	A	A

**Execution** (*Smem* (15 – T(3–0))) → TC

**Status Bits** Affects TC

**Description** This instruction copies the specified bit of the data-memory value *Smem* into the TC bit in status register ST0. The four LSBs of T contain a bit code that specifies which bit is copied.

The bit address corresponds to (15 – T(3–0)). The bit code corresponds to the content of T(3–0).

**Bit Codes for This Instruction**

Bit Address		Bit Code	Bit Address		Bit Code
(LSB)	0	1111	8	0111	
	1	1110	9	0110	
	2	1101	10	0101	
	3	1100	11	0100	
	4	1011	12	0011	
	5	1010	13	0010	
	6	1001	14	0001	
	7	1000	(MSB) 15	0000	

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 3A (see page 3-5)  
Class 3B (see page 3-6)

## BITT *Test Bit Specified by T*

---

### Example

BITT \*AR7+0

	Before Instruction	After Instruction
T	<input type="text" value="C"/>	<input type="text" value="C"/>
TC	<input type="text" value="0"/>	<input type="text" value="1"/>
AR0	<input type="text" value="0008"/>	<input type="text" value="0008"/>
AR7	<input type="text" value="0100"/>	<input type="text" value="0108"/>
Data Memory		
0100h	<input type="text" value="0008"/>	<input type="text" value="0008"/>



**Syntax** **CALA[D] src**

**Operands** src: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	S	1	1	1	0	0	0	1	1

**Execution** **Nondelayed**  
 $(SP) - 1 \rightarrow SP$   
 $(PC) + 1 \rightarrow TOS$   
 $(src(15-0)) \rightarrow PC$

**Delayed**  
 $(SP) - 1 \rightarrow SP$   
 $(PC) + 3 \rightarrow TOS$   
 $(src(15-0)) \rightarrow PC$

**Status Bits** None

**Description** This instruction passes control to the 16-bit address in the low part of *src* (bits 15–0). If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 6 cycles  
4 cycles (delayed)

**Classes** Class 30B (see page 3-67)

**Example 1** CALA A

	Before Instruction	After Instruction
A	00 0000 3000	00 0000 3000
PC	0025	3000
SP	1111	1110
Data Memory		
1110h	4567	0026

**CALA[D]** *Call Subroutine at Location Specified by Accumulator*

---

**Example 2**

CALAD B

ANDM 4444h, \*AR1+

	Before Instruction	After Instruction
B	00 0000 2000	00 0000 2000
PC	0025	2000
SP	1111	1110
Data Memory		
1110h	4567	0028

After the memory location has been ANDed with 4444h, the program continues executing from location 2000h.

**Syntax** **CALL[D] *pmad***

**Operands**  $0 \leq pmad \leq 65\,535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	Z	0	0	1	1	1	0	1	0	0
16-bit constant															

**Execution**

**Nondelayed**  
 $(SP) - 1 \rightarrow SP$   
 $(PC) + 2 \rightarrow TOS$   
 $pmad \rightarrow PC$

**Delayed**  
 $(SP) - 1 \rightarrow SP$   
 $(PC) + 4 \rightarrow TOS$   
 $pmad \rightarrow PC$

**Status Bits** None

**Description** This instruction passes control to the specified program-memory address (*pmad*). The return address is pushed onto the TOS before *pmad* is loaded into PC. If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words** 2 words

**Cycles** 4 cycles  
2 cycles (delayed)

**Classes** Class 29B (see page 3-66)

## **CALL[D]** *Call Unconditionally*

---

### **Example 1**

CALL 3333h

	Before Instruction	After Instruction
PC	0025	3333
SP	1111	1110
Data Memory		
1110h	4567	0027

### **Example 2**

CALLD 1000h  
ANDM \*AR1+, 4444h

	Before Instruction	After Instruction
PC	0025	1000
SP	1111	1110
Data Memory		
1110h	4567	0029

**Syntax** **CC[D]** *pmad*, *cond* [, *cond* [, *cond* ]]

**Operands**  $0 \leq \text{pmad} \leq 65\,535$

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{\text{BIO}}$ low	0000 0011	NBIO	$\overline{\text{BIO}}$ high	0000 0010
C	$C = 1$	0000 1100	NC	$C = 0$	0000 1000
TC	$\text{TC} = 1$	0011 0100	NTC	$\text{TC} = 0$	0010 0000
AEQ	$(A) = 0$	0100 0101	BEQ	$(B) = 0$	0100 1101
ANEQ	$(A) \neq 0$	0100 0100	BNEQ	$(B) \neq 0$	0100 1100
AGT	$(A) > 0$	0100 0110	BGT	$(B) > 0$	0100 1110
AGEQ	$(A) \geq 0$	0100 0010	BGEQ	$(B) \geq 0$	0100 1010
ALT	$(A) < 0$	0100 0011	BLT	$(B) < 0$	0100 1011
ALEQ	$(A) \leq 0$	0100 0111	BLEQ	$(B) \leq 0$	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	Z	1	C	C	C	C	C	C	C	C
16-bit constant															

**Execution**

**Nondelayed**

If (cond(s))

Then

$(\text{SP}) - 1 \rightarrow \text{SP}$

$(\text{PC}) + 2 \rightarrow \text{TOS}$

$\text{pmad} \rightarrow \text{PC}$

Else

$(\text{PC}) + 2 \rightarrow \text{PC}$

## CC[D] Call Conditionally

---

### Delayed

If (cond(s))

Then

(SP) - 1 → SP

(PC) + 4 → TOS

p<sub>mad</sub> → PC

Else

(PC) + 2 → PC

### Status Bits

Affects OVA or OVB (if OV or NOV is chosen)

### Description

This instruction passes control to the program-memory address (*p<sub>mad</sub>*) if the specified condition(s) is met. If the condition(s) is not met, the PC is incremented by 2. If the call is delayed (specified by the D suffix), and these words have no effect on the condition tested, the two 1-word instructions or the one 2-word instruction following the call is fetched from program memory and executed.

This instruction tests multiple conditions before passing control to another section of the program. This instruction can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

Group 1: You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time.

Group 2: You can select up to three conditions. Each of these conditions must be from a different category (category A, B, or C); you cannot have two conditions from the same category. For example, you can test TC, C, and BIO at the same time but you cannot test NTC, C, and NC at the same time.

## Conditions for This Instruction

Group 1		Group 2		
Category A	Category B	Category A	Category B	Category C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

**Note:**

This instruction is not repeatable.

**Words**

2 words

**Cycles**

5 cycles (true condition)  
 3 cycles (false condition)  
 3 cycles (delayed)

**Classes**

Class 31B (see page 3-69)

**Example 1**

CC 2222h, AGT

Before Instruction		After Instruction	
A	00 0000 3000	A	00 0000 3000
PC	0025	PC	2222
SP	1111	SP	1110
Data Memory			
1110h	4567	1110h	0027

**Example 2**

CCD 1000h, BOV  
 ANDM 4444h, \*AR1+

Before Instruction		After Instruction	
PC	0025	PC	1000
OV B	1	OV B	0
SP	1111	SP	1110
Data Memory			
1110h	4567	1110h	0029

## CMPL *Complement Accumulator*

**Syntax** CMPL *src* [, *dst*]

**Operands** src, dst: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	1	0	0	1	1

**Execution**  $\overline{(\text{src})} \rightarrow \text{dst}$

**Status Bits** None

**Description** This instruction calculates the 1s complement of the content of *src* (this is a logical inversion). The result is stored in *dst*, if specified, or *src* otherwise.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example** CMPL A, B

	Before Instruction	After Instruction
A	FC DFFA AEAA	FC DFFA AEAA
B	00 0000 7899	03 2005 5155



**Syntax** **CMPM** *Smem*, #*lk***Operands** *Smem*: Single data-memory operand  
 $-32\,768 \leq lk \leq 32\,767$ **Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	1	A	A	A	A	A	A	A
16-bit constant															

**Execution** If (*Smem*) = *lk*  
Then  
1 → TC  
Else  
0 → TC**Status Bits** Affects TC**Description** This instruction compares the 16-bit single data-memory operand *Smem* to the 16-bit constant *lk*. If they are equal, TC is set to 1. Otherwise, TC is cleared to 0.**Words** 2 wordsAdd 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.**Cycles** 2 cyclesAdd 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.**Classes** Class 6A (see page 3-10)  
Class 6B (see page 3-11)**Example** **CMPM** \*AR4+, 0404h

	Before Instruction	After Instruction
TC	1	0
AR4	0100	0101
Data Memory		
0100h	4444	4444

## **CMPR** *Compare Auxiliary Register With AR0*

**Syntax** **CMPR** *CC*, *ARx*

**Operands**  $0 \leq CC \leq 3$   
*ARx*: AR0–AR7

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	C	C	1	0	1	0	1	A	R	X

**Execution** If (cond)  
Then  
    1 → TC  
Else  
    0 → TC

**Status Bits** Affects TC

**Description** This instruction compares the content of the designated auxiliary register (*ARx*) to the content of AR0 and sets the TC bit according to the comparison. The comparison is specified by the *CC* (condition code) value (see the following table). If the condition is true, TC is set to 1. If the condition is false, TC is cleared to 0. All conditions are computed as unsigned operations.

Condition	Condition Code (CC)	Description
EQ	00	Test if (ARx) = (AR0)
LT	01	Test if (ARx) < (AR0)
GT	10	Test if (ARx) > (AR0)
NEQ	11	Test if (ARx) ≠ (AR0)

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example** CMPR 2, AR4

	Before Instruction	After Instruction
TC	1	0
AR0	FFFF	FFFF
AR4	7FFF	7FFF

Syntax	CMPS src, Smem																																
Operands	src: A (accumulator A) B (accumulator B) Smem: Single data-memory operand																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	0	0	1	1	1	S	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	0	0	0	1	1	1	S	I	A	A	A	A	A	A	A																		
Execution	If ((src(31–16)) > (src(15–0))) Then (src(31–16)) → Smem (TRN) << 1 → TRN 0 → TRN(0) 0 → TC Else (src(15–0)) → Smem (TRN) << 1 → TRN 1 → TRN(0) 1 → TC																																
Status Bits	Affects TC																																
Description	<p>This instruction compares the two 16-bit 2s-complement values located in the high and low parts of <i>src</i> and stores the maximum value in the single data-memory location <i>Smem</i>. If the high part of <i>src</i> (bits 31–16) is greater, a 0 is shifted into the LSB of the transition (TRN) register and the TC bit is cleared to 0. If the low part of <i>src</i> (bits 15–0) is greater, a 1 is shifted into the LSB of the TRN register and the TC bit is set to 1.</p> <p>This instruction does not follow the standard pipeline operation. The comparison is performed in the read phase; thus, the <i>src</i> value is the value one cycle before the instruction executes. The TRN register and the TC bit are updated during the execution phase.</p>																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Classes	Class 10A (see page 3-22) Class 10B (see page 3-23)																																

## CMPS *Compare, Select and Store Maximum*

---

### Example

CMPS A, \*AR4+

	Before Instruction	After Instruction
A	00 2345 7899	00 2345 7899
TC	0	1
AR4	0100	0101
TRN	4444	8889
Data Memory		
0100h	0000	7899

**Syntax** **DADD** *Lmem*, *src* [, *dst*]

**Operands**  
*Lmem*: Long data-memory operand  
*src*, *dst*: A (accumulator A)  
               B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	S	D	I	A	A	A	A	A	A	A

**Execution**  
 If C16 = 0  
 Then  
     (*Lmem*) + (*src*) → *dst*  
 Else  
     (*Lmem*(31–16)) + (*src*(31–16)) → *dst*(39–16)  
     (*Lmem*(15–0)) + (*src*(15–0)) → *dst*(15–0)

**Status Bits**  
 Affected by SXM and OVM (only if C16 = 0)  
 Affects C and OVdst (or OVsrc, if *dst* is not specified)

**Description**  
 This instruction adds the content of *src* to the 32-bit long data-memory operand *Lmem*. If a *dst* is specified, this instruction stores the result in *dst*. If no *dst* is specified, this instruction stores the result in *src*. The value of C16 determines the mode of the instruction:

- ☐ If C16 = 0, the instruction is executed in double-precision mode. The 40-bit *src* value is added to the *Lmem*. The saturation and overflow bits are set according to the result of the operation.
- ☐ If C16 = 1, the instruction is executed in dual 16-bit mode. The high part of *src* (bits 31–16) is added to the 16 MSBs of *Lmem*, and the low part of *src* (bits 15–0) is added to the 16 LSBs of *Lmem*. The saturation and overflow bits are not affected in this mode. In this mode, the results are not saturated regardless of the state of the OVM bit.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes**  
 Class 9A (see page 3-20)  
 Class 9B (see page 3-21)

## DADD Double-Precision/Dual 16-Bit Add to Accumulator

### Example 1

DADD \*AR3+, A, B

	Before Instruction	After Instruction
A	00 5678 8933	00 5678 8933
B	00 0000 0000	00 6BAC BD89
C16	0	0
AR3	0100	AR3† 0102
Data Memory		
0100h	1534	0100h 1534
0101h	3456	0101h 3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

### Example 2

DADD \*AR3-, A, B

	Before Instruction	After Instruction
A	00 5678 3933	A 00 5678 3933
B	00 0000 0000	B 00 6BAC 6D89
C16	1	C16 1
AR3	0100	AR3† 00FE
Data Memory		
0100h	1534	0100h 1534
0101h	3456	0101h 3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

### Example 3

DADD \*AR3-, A, B

	Before Instruction	After Instruction
A	00 5678 3933	A 00 5678 3933
B	00 0000 0000	B 00 8ACE 4E67
C16	0	C16 0
AR3	0101	AR3† 0103
Data Memory		
0100h	1534	0100h 1534
0101h	3456	0101h 3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax	DADST Lmem, dst																																
Operands	Lmem: Long data-memory operand dst: A (accumulator A) B (accumulator B)																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>D</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	1	1	0	1	D	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	0	1	1	0	1	D	I	A	A	A	A	A	A	A																		
Execution	If C16 = 1 Then (Lmem(31-16)) + (T) → dst(39-16) (Lmem(15-0)) - (T) → dst(15-0) Else (Lmem) + ((T) + (T) << 16) → dst																																
Status Bits	Affected by SXM and OVM (only if C16 = 0) Affects C and OVDst																																
Description	<p>This instruction adds the content of T to the 32-bit long data-memory operand <i>Lmem</i>. The value of C16 determines the mode of the instruction:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> If C16 = 0, the instruction is executed in double-precision mode. <i>Lmem</i> is added to a 32-bit value composed of the content of T concatenated with the content of T left-shifted 16 bits (T &lt;&lt; 16 + T). The result is stored in <i>dst</i>.</li><li><input type="checkbox"/> If C16 = 1, the instruction is executed in dual 16-bit mode. The 16 MSBs of the <i>Lmem</i> are added to the content of T and stored in the upper 24 bits of <i>dst</i>. At the same time, the content of T is subtracted from the 16 LSBs of <i>Lmem</i>. The result is stored in the lower 16 bits of <i>dst</i>. In this mode, the results are not saturated regardless of the state of the OVM bit.</li></ul> <div><b>Note:</b> This instruction is meaningful only if C16 is set to 1 (dual 16-bit mode).</div>																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.																																
Classes	Class 9A (see page 3-20) Class 9B (see page 3-21)																																

## **DADST** *Double-Precision Load With T Add/Dual 16-Bit Load With T Add/Subtract*

### **Example 1**

DADST \*AR3-, A

Before Instruction		After Instruction	
A	00 0000 0000	A	00 3879 1111
T	2345	T	2345
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

### **Example 2**

DADST \*AR3+, A

Before Instruction		After Instruction	
A	00 0000 0000	A	00 3879 579B
T	2345	T	2345
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.



**Syntax** **DELAY** *Smem***Operands** *Smem*: Single data-memory operand
**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	1	I	A	A	A	A	A	A	A

**Execution** (*Smem*) → *Smem* + 1**Status Bits** None

**Description** This instruction copies the content of a single data-memory location *Smem* into the next higher address. When data is copied, the content of the addressed location remains the same. This function is useful for implementing a Z delay in digital signal processing applications. The delay operation is also contained in the load T and insert delay (LTD) instruction (page 4-81) and the multiply by program memory and accumulate with delay (MACD) instruction (page 4-87).

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 24A (see page 3-56)  
Class 24B (see page 3-56)

**Example** **DELAY** \*AR3

	Before Instruction		After Instruction		
AR3	<table><tr><td>0100</td></tr></table>	0100	AR3	<table><tr><td>0100</td></tr></table>	0100
0100					
0100					
Data Memory					
0100h	<table><tr><td>6CAC</td></tr></table>	6CAC	0100h	<table><tr><td>6CAC</td></tr></table>	6CAC
6CAC					
6CAC					
0101h	<table><tr><td>0000</td></tr></table>	0000	0101h	<table><tr><td>6CAC</td></tr></table>	6CAC
0000					
6CAC					

## DLD Double-Precision/Dual 16-Bit Long-Word Load to Accumulator

Syntax	<b>DLD</b> <i>Lmem</i> , <i>dst</i>																																
Operands	<i>Lmem</i> : Long data-memory operand <i>dst</i> : A (accumulator A) B (accumulator B)																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>D</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	1	0	1	1	D	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	0	1	0	1	1	D	I	A	A	A	A	A	A	A																		
Execution	If C16 = 0 Then ( <i>Lmem</i> ) → <i>dst</i> Else ( <i>Lmem</i> (31–16)) → <i>dst</i> (39–16) ( <i>Lmem</i> (15–0)) → <i>dst</i> (15–0)																																
Status Bits	Affected by SXM																																
Description	This instruction loads <i>dst</i> with a 32-bit long operand <i>Lmem</i> . The value of C16 determines the mode of the instruction:  <input type="checkbox"/> If C16 = 0, the instruction is executed in double-precision mode. <i>Lmem</i> is loaded to <i>dst</i> .  <input type="checkbox"/> If C16 = 1, the instruction is executed in dual 16-bit mode. The 16 MSBs of <i>Lmem</i> are loaded to the upper 24 bits of <i>dst</i> . At the same time, the 16 LSBs of <i>Lmem</i> are loaded in the lower 16 bits of <i>dst</i> .																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Classes	Class 9A (see page 3-20) Class 9B (see page 3-21)																																
Example	DLD *AR3+, B																																

	Before Instruction	After Instruction
B	00 0000 0000	00 6CAC BD90
AR3	0100	AR3† 0102
Data Memory		
0100h	6CAC	0100h 6CAC
0101h	BD90	0101h BD90

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

Syntax	DRSUB <i>Lmem</i> , <i>src</i>																																
Operands	<div>Lmem: Long data-memory operand</div> <div>src: A (accumulator A) B (accumulator B)</div>																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	1	1	0	0	S	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	0	1	1	0	0	S	I	A	A	A	A	A	A	A																		
Execution	<div>If C16 = 0</div> <div>Then</div> <div>(Lmem) − (src) → src</div> <div>Else</div> <div>(Lmem(31–16)) − (src(31–16)) → src(39–16)</div> <div>(Lmem(15–0)) − (src(15–0)) → src(15–0)</div>																																
Status Bits	<div>Affected by SXM and OVM (only if C16 = 0)</div> <div>Affects C and OVsrc</div>																																
Description	<div>This instruction subtracts the content of <i>src</i> from the 32-bit long data-memory operand <i>Lmem</i> and stores the result in <i>src</i>. The value of C16 determines the mode of the instruction:</div> <div><div><input type="checkbox"/> If C16 = 0, the instruction is executed in double-precision mode. The content of <i>src</i> (32 bits) is subtracted from <i>Lmem</i>. The result is stored in <i>src</i>.</div><div><input type="checkbox"/> If C16 = 1, the instruction is executed in dual 16-bit mode. The high part of <i>src</i> (bits 31–16) is subtracted from the 16 MSBs of <i>Lmem</i> and the result is stored in the high part of <i>src</i> (bits 39–16). At the same time, the low part of <i>src</i> (bits 15–0) is subtracted from the 16 LSBs of <i>Lmem</i>. The result is stored in the low part of <i>src</i> (bits 15–0). In this mode, the results are not saturated regardless of the state of the OVM bit.</div></div>																																
Words	<div>1 word</div> <div>Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.</div>																																
Cycles	<div>1 cycle</div> <div>Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.</div>																																
Classes	<div>Class 9A (see page 3-20)</div> <div>Class 9B (see page 3-21)</div>																																

## DRSUB Double-Precision/Dual 16-Bit Subtract From Long Word

### Example 1

DRSUB \*AR3+, A

Before Instruction		After Instruction	
A	00 5678 8933	A	FF BEBB AB23
C	x	C	0
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

### Example 2

DRSUB \*AR3-, A

Before Instruction		After Instruction	
A	00 5678 3933	A	FF BEBC FB23
C	1	C	0
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax	DSADT <i>Lmem</i> , <i>dst</i>																																
Operands	<i>Lmem</i> : Long data-memory operand <i>dst</i> : A (accumulator A) B (accumulator B)																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>D</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	1	1	1	1	D	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	0	1	1	1	1	D	I	A	A	A	A	A	A	A																		
Execution	If C16 = 1 Then ( <i>Lmem</i> (31–16)) – (T) → <i>dst</i> (39–16) ( <i>Lmem</i> (15–0)) + (T) → <i>dst</i> (15–0) Else ( <i>Lmem</i> ) – ((T) + (T << 16)) → <i>dst</i>																																
Status Bits	Affected by SXM and OVM (only if C16 = 0) Affects C and OV <i>dst</i>																																
Description	<p>This instruction subtracts/adds the content of T from the 32-bit long data-memory operand <i>Lmem</i> and stores the result in <i>dst</i>. The value of C16 determines the mode of the instruction:</p> <div><div><input type="checkbox"/> If C16 = 0, the instruction is executed in double-precision mode. A 32-bit value composed of the content of T concatenated with the content of T left-shifted 16 bits (T &lt;&lt; 16 + T) is subtracted from <i>Lmem</i>. The result is stored in <i>dst</i>.</div><div><input type="checkbox"/> If C16 = 1, the instruction is executed in dual 16-bit mode. The content of T is subtracted from the 16 MSBs of <i>Lmem</i> and the result is stored in the high part of <i>dst</i> (bits 39–16). At the same time, the content of T is added to the 16 LSBs of <i>Lmem</i> and the result is stored in the low part of <i>dst</i> (bits 15–0). In this mode, the results are not saturated regardless of the state of the OVM bit.</div></div> <div><div>Note:</div><div>This instruction is meaningful only if C16 is set (dual 16-bit mode).</div></div>																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																

## DSADT Long-Word Load With T Add/Dual 16-Bit Load With T Subtract/Add

**Classes** Class 9A (see page 3-20)  
Class 9B (see page 3-21)

### Example 1

DSADT \*AR3+, A

Before Instruction		After Instruction	
A	00 0000 0000	A	FF F1EF 1111
T	2345	T	2345
C	0	C	1
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

### Example 2

DSADT \*AR3-, A

Before Instruction		After Instruction	
A	00 0000 0000	A	FF F1EF 579B
T	2345	T	2345
C	0	C	1
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

Syntax	DST src, Lmem																																
Operands	src: A (accumulator A) B (accumulator B) Lmem: Long data-memory operand																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	1	1	1	S	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	0	0	1	1	1	S	I	A	A	A	A	A	A	A																		
Execution	(src(31–0)) → Lmem																																
Status Bits	None																																
Description	This instruction stores the content of src in a 32-bit long data-memory location Lmem.																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.																																

<b>Cycles</b>	2 cycles  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.
<b>Classes</b>	Class 13A (see page 3-28) Class 13B (see page 3-29)

**Example 1**

DST B, \*AR3+

	Before Instruction	After Instruction
B	00 6CAC BD90	00 6CAC BD90
AR3	0100	AR3† 0102
Data Memory		
0100h	0000	0100h 6CAC
0101h	0000	0101h BD90

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

**Example 2**

DST B, \*AR3–

	Before Instruction	After Instruction
B	00 6CAC BD90	B 00 6CAC BD90
AR3	0101	AR3† 00FF
Data Memory		
0100h	0000	0100h BD90
0101h	0000	0101h 6CAC

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

## DSUB Double-Precision/Dual 16-Bit Subtract From Accumulator

**Syntax** DSUB *Lmem*, *src*

**Operands** Lmem: Long data-memory operand  
src: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	S	I	A	A	A	A	A	A	A

**Execution** If C16 = 0  
Then  
    (src) – (Lmem) → src  
Else  
    (src(31–16)) – (Lmem(31–16)) → src(39–16)  
    (src(15–0)) – (Lmem(15–0)) → src(15–0)

**Status Bits** Affected by SXM and OVM (only if C16 = 0)  
Affects C and OVsrc

**Description** This instruction subtracts the 32-bit long data-memory operand *Lmem* from the content of *src*, and stores the result in *src*. The value of C16 determines the mode of the instruction:

- ☐ If C16 = 0, the instruction is executed in double-precision mode. *Lmem* is subtracted from the content of *src*.
- ☐ If C16 = 1, the instruction is executed in dual 16-bit mode. The 16 MSBs of *Lmem* are subtracted from the high part of *src* (bits 31–16) and the result is stored in the high part of *src* (bits 39–16). At the same time, the 16 LSBs of *Lmem* are subtracted from the low part of *src* (bits 15–0) and the result is stored in the low part of *src* (bits 15–0).

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 9A (see page 3-20)  
Class 9B (see page 3-21)



## Double-Precision/Dual 16-Bit Subtract From Accumulator **DSUB**

### Example 1

DSUB \*AR3+, A

	Before Instruction	After Instruction
A	00 5678 8933	00 4144 54DD
C16	0	0
AR3	0100	AR3† 0102
Data Memory		
0100h	1534	0100h 1534
0101h	3456	0101h 3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

### Example 2

DSUB \*AR3-, A

	Before Instruction	After Instruction
A	00 5678 3933	00 4144 04DD
C	1	0
C16	1	1
AR3	0100	AR3† 00FE
Data Memory		
0100h	1534	0100h 1534
0101h	3456	0101h 3456

† Because this instruction is a long-operand instruction, AR3 is decremented by 2 after the execution.

**DSUBT** *Long-Word Load With T Subtract/Dual 16-Bit Load With T Subtract***Syntax** **DSUBT** *Lmem, dst***Operands**  
**Lmem:** Long data-memory operand  
**dst:** A (accumulator A)  
B (accumulator B)**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	D	I	A	A	A	A	A	A	A

**Execution**  
If C16 = 1  
Then  
     $(Lmem(31-16)) - (T) \rightarrow dst(39-16)$   
     $(Lmem(15-0)) - (T) \rightarrow dst(15-0)$   
Else  
     $(Lmem) - ((T) + (T \ll 16)) \rightarrow dst$ **Status Bits**  
Affected by SXM and OVM (only if C16 = 0)  
Affects C and OVDst**Description**  
This instruction subtracts the content of T from the 32-bit long data-memory operand *Lmem* and stores the result in *dst*. The value of C16 determines the mode of the instruction:

- ☐ If C16 = 0, the instruction is executed in double-precision mode. A 32-bit value composed of the content of T concatenated with the content of T left-shifted 16 bits ( $T \ll 16 + T$ ) is subtracted from *Lmem*. The result is stored in *dst*.
- ☐ If C16 = 1, the instruction is executed in dual 16-bit mode. The content of T is subtracted from the 16 MSBs of *Lmem* and the result is stored in the high part of *dst* (bits 39–16). At the same time, the content of T is subtracted from the 16 LSBs of *Lmem* and the result is stored in the low part of *dst* (bits 15–0). In this mode, the results are not saturated regardless of the value of the OVM bit.

**Note:**

This instruction is meaningful only if C16 is set to 1 (dual 16-bit mode).

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes**                      Class 9A (see page 3-20)  
                                      Class 9B (see page 3-21)

**Example 1**

DSUBT \*AR3+, A

	Before Instruction		After Instruction
A	00 0000 0000	A	FF F1EF 1111
T	2345	T	2345
C16	0	C16	0
AR3	0100	AR3†	0102
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long-operand instruction, AR3 is incremented by 2 after the execution.

**Example 2**

DSUBT \*AR3-, A

	Before Instruction		After Instruction
A	00 0000 0000	A	FF F1EF 1111
T	2345	T	2345
C16	1	C16	1
AR3	0100	AR3†	00FE
Data Memory			
0100h	1534	0100h	1534
0101h	3456	0101h	3456

† Because this instruction is a long operand instruction, AR3 is decremented by 2 after the execution.

## EXP Accumulator Exponent

**Syntax** EXP *src*

**Operands** src: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	S	1	0	0	0	1	1	1	0

**Execution**

If (src) = 0  
Then  
0 → T  
Else  
(Number of leading bits of src) – 8 → T

**Status Bits** None

**Description** This instruction computes the exponent value, which is a signed 2s-complement value in the –8 to 31 range, and stores the result in T. The exponent is computed by calculating the number of leading bits in *src* and subtracting 8 from this value. The number of leading bits is equivalent to the number of left shifts needed to eliminate the significant bits from the 40-bit *src* with the exception of the sign bit. The *src* is not modified after this instruction.

The result of subtracting 8 from the number of leading bits produces a negative exponent for accumulator values that have significant bits in the guard bits (the eight MSBs of the accumulator used in error detection and correction). See the normalization instruction (page 4-122).

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1** EXP A

Before Instruction			After Instruction		
A	FF FFFF FFCB	-53	A	FF FFFF FFCB	-53
T	0000		T	0019	25

**Example 2** EXP B

Before Instruction			After Instruction		
B	07 8543 2105		B	07 8543 2105	
T	FFFC		T	FFFC	

-4†

† The value in accumulator B has significant bits in the guard bits, which results in a negative exponent.

**Syntax** **FB[D]** *extpmad***Operands**  $0 \leq \text{extpmad} \leq 7F\ FFFF$ 

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	0	Z	0	1	7-bit constant = p <sub>mad</sub> (22–16)						
	16-bit constant = p <sub>mad</sub> (15–0)															

**Execution** (*pmad*(15–0)) → PC  
 (*pmad*(22–16)) → XPC

**Status Bits** None

**Description** This instruction passes control to the program-memory address *pmad* (bits 15–0) on the page specified by *pmad* (bits 22–16). The *pmad* can be either a symbolic or numeric address. If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words** 2 words

**Cycles** 4 cycles  
 2 cycles (delayed)

**Classes** Class 29A (see page 3-66)**Example 1** FB 012000h

	Before Instruction		After Instruction
PC	1000	PC	2000
XPC	00	XPC	01

2000h is loaded into the PC, 01h is loaded into XPC, and the program continues executing from that location.

**Example 2** FBD 7F1000h  
 ANDM #4444h, \*AR1+

	Before Instruction		After Instruction
PC	2000	PC	1000
XPC	00	XPC	7F

After the operand has been ANDed with 4444h, the program continues executing from location 1000h on page 7Fh.

## **FBACC[D]** *Far Branch to Location Specified by Accumulator ('548)*

**Syntax** **FBACC[D]** *src*

**Operands** *src*: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	S	1	1	1	0	0	1	1	0

**Execution** (*src*(15–0)) → PC  
(*src*(22–16)) → XPC

**Status Bits** None

**Description** This instruction loads the XPC with the value in *src* (bits 22–16) and passes control to the 16-bit address in the low part of *src* (bits 15–0). If the branch is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the branch instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 6 cycles  
4 cycles (delayed)

**Classes** Class 30A (see page 3-67)

**Example 1** FBACC A

	Before Instruction		After Instruction
A	00 0001 3000	A	00 0001 3000
PC	1000	PC	3000
XPC	00	XPC	01

1h is loaded into the XPC, 3000h is loaded into the PC, and the program continues executing from that location on page 1h.

**Example 2** FBACCD B

ANDM 4444h \*AR1+

	Before Instruction		After Instruction
B	00 007F 2000	B	00 007F 2000
XPC	01	XPC	7F

After the operand has been ANDed with 4444h value, 7Fh is loaded into the XPC, and the program continues executing from location 2000h on page 7Fh.

*Far Call Subroutine at Location Specified by Accumulator ('548)* **FCALA[D]**

**Syntax**                    **FCALA[D] src**

**Operands**                src:    A (accumulator A)  
                                      B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	S	1	1	1	0	0	1	1	1

**Execution**                **Nondelayed**  
                                   (SP) – 1 → SP  
                                   (PC) + 1 → TOS  
                                   (SP) – 1 → SP  
                                   (XPC) → TOS  
                                   (src(15–0)) → PC  
                                   (src(22–16)) → XPC

**Delayed**  
                                   (SP) – 1 → SP  
                                   (PC) + 3 → TOS  
                                   (SP) – 1 → SP  
                                   (XPC) → TOS  
                                   (src(15–0)) → PC  
                                   (src(22–16)) → XPC

**Status Bits**                None

**Description**                This instruction loads the XPC with the value in *src* (bits 22–16) and passes control to the 16-bit address in the low part of *src* (bits 15–0). If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words**                      1 word

**Cycles**                      6 cycles  
                                   4 cycles (delayed)

**Classes**                    Class 30B (see page 3-67)

## **FCALA[D]** *Far Call Subroutine at Location Specified by Accumulator ('548)*

### **Example 1**

FCALA A

Before Instruction		After Instruction	
A	00 007F 3000	A	00 007F 3000
PC	0025	PC	3000
XPC	00	XPC	7F
SP	1111	SP	1109
Data Memory			
1110h	4567	1110h	0026
1109h	4567	1109h	0000

### **Example 2**

FCALAD B

ANDM 4444h+

Before Instruction		After Instruction	
B	00 0020 2000	B	00 0020 2000
PC	0025	PC	3000
XPC	7F	XPC	20
SP	1111	SP	1109
Data Memory			
1110h	4567	1110h	0028
1109h	4567	1109h	007F

After the memory location has been ANDed with 4444h, the program continues executing from location 2000h on page 20h.



**Syntax** FCALL[D] *extpmad*

**Operands**  $0 \leq \text{extpmad} \leq 7F\text{ FFFF}$

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	0	Z	1	1	7-bit constant = pmad(22-16)						
	16-bit constant = pmad(15-0)															

**Execution**

**Nondelayed**

(SP) – 1 → SP  
(PC) + 2 → TOS  
(SP) – 1 → SP  
(XPC) → TOS  
(*pmad*(15–0)) → PC  
(*pmad*(22–16)) → XPC

**Delayed**

(SP) – 1 → SP  
(PC) + 4 → TOS  
(SP) – 1 → SP  
(XPC) → TOS  
(*pmad*(15–0)) → PC  
(*pmad*(22–16)) → XPC

**Status Bits** None

**Description** This instruction passes control to the specified program-memory address *pmad* (bits 15–0) on the page specified by *pmad* (bits 22–16). The return address is pushed onto the stack before *pmad* is loaded into PC. If the call is delayed (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following the call instruction is fetched from program memory and executed.

**Note:**

This instruction is not repeatable.

**Words** 2 words

**Cycles** 4 cycles  
2 cycles (delayed)

**Classes** Class 29B (see page 3-66)

## FCALL[D] *Far Call Unconditionally ('548)*

### Example 1

FCALL 013333h

Before Instruction		After Instruction	
PC	0025	PC	3333
XPC	00	XPC	01
SP	1111	SP	1109
Data Memory			
1110h	4567	1110h	0027
1109h	4567	1109h	0000

### Example 2

FCALLD 301000h

ANDM #4444h

Before Instruction		After Instruction	
PC	3001	PC	1000
XPC	7F	XPC	30
SP	1111	SP	1109
Data Memory			
1110h	4567	1110h	3001
1109h	4567	1109h	007F

**Syntax** **FIRS** *Xmem, Ymem, pmad***Operands** *Xmem, Ymem*: Dual data-memory operands  
 $0 \leq pmad \leq 65535$ **Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	0	X	X	X	X	Y	Y	Y	Y
16-bit constant															

**Execution***pmad* → PAR

While (RC) ≠ 0

 $(B) + (A(32-16)) \times (Pmem \text{ addressed by } PAR) \rightarrow B$  $((Xmem) + (Ymem)) \ll 16 \rightarrow A$  $(PAR) + 1 \rightarrow PAR$  $(RC) - 1 \rightarrow RC$ **Status Bits**

Affected by SXM, FRCT, and OVM

Affects C, OVA, and OVB

**Description**

This instruction implements a symmetrical finite impulse response (FIR) filter. This instruction multiplies accumulator A (bits 32–16) with a Pmem value addressed by *pmad* (in the program address register PAR) and adds the result to the value in accumulator B. At the same time, it adds the memory operands *Xmem* and *Ymem*, shifts the result left 16 bits, and loads this value into accumulator A. In the next iteration, *pmad* is incremented by 1. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words**

2 words

**Cycles**

3 cycles

**Classes**

Class 8 (see page 3-15)

**Example****FIRS** \*AR3+, \*AR4+, COEFS

Before Instruction		After Instruction	
A	00 0077 0000	A	00 00FF 0000
B	00 0000 0000	B	00 0008 762C
FRCT	0	FRCT	0
AR3	0100	AR3	0101
AR4	0200	AR4	0201
Data Memory			
0100h	0055	0100h	0055
0200h	00AA	0200h	00AA
Program Memory			
COEFS	1234	COEFS	1234

## FRAME *Stack Pointer Immediate Offset*

**Syntax** FRAME *K*

**Operands**  $-128 \leq K \leq 127$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	1	0	K	K	K	K	K	K	K	K

**Execution**  $(SP) + K \rightarrow SP$

**Status Bits** None

**Description** This instruction adds a short-immediate offset *K* to the SP. There is no latency for address generation in compiler mode (CPL = 1) or for stack manipulation by the instruction following this instruction.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example** FRAME 10h

Before Instruction		After Instruction	
SP	1000	SP	1010

**Syntax** FRET[D]

**Operands** None

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	0	1	1	1	0	0	1	0	0

**Execution**

(TOS) → XPC  
 (SP) + 1 → SP  
 (TOS) → PC  
 (SP) + 1 → SP

**Status Bits** None

**Description** This instruction replaces the XPC with the 7-bit value from the TOS and replaces the PC with the next 16-bit value on the stack. The SP is incremented by 1 for each of the two replacements. If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 6 cycles  
 4 cycles (delayed)

**Classes** Class 34 (see page 3-71)

**Example** FRET

	Before Instruction	After Instruction
PC	2112	1000
XPC	01	05
SP	0300	0302
Data Memory		
0300h	0005	0005
0301h	1000	1000

**FRETE[D]** *Enable Interrupts and Far Return From Interrupt ('548)***Syntax** FRETE[D]**Operands** None**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	0	1	1	1	0	0	1	0	1

**Execution**  
(TOS) → XPC  
(SP) + 1 → SP  
(TOS) → PC  
(SP) + 1 → SP  
0 → INTM**Status Bits** Affects INTM**Description** This instruction replaces the XPC with the 7-bit value from the TOS and replaces the PC with the next 16-bit value on the stack, continuing execution from the new PC value. This instruction automatically clears the interrupt mask bit (INTM) in ST1. (Clearing this bit enables interrupts.) If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.**Note:**

This instruction is not repeatable.

**Words** 1 word**Cycles** 6 cycles  
4 cycles (delayed)**Classes** Class 34 (see page 3-71)**Example** FRETE

	Before Instruction	After Instruction
PC	2112	0110
XPC	05	6E
ST1	xCxx	x4xx
SP	0300	0302
Data Memory		
0300h	006E	006E
0301h	0110	0110

**Syntax** IDLE K

**Operands**  $1 \leq K \leq 3$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	N	N	1	1	1	0	0	0	0	1

If K is:	NN is:
1	00
2	10
3	01

**Execution** (PC) +1 → PC

**Status Bits** Affected by INTM

**Description**

This instruction forces the program being executed to wait until an unmasked interrupt or reset occurs. The PC is incremented by 1. The device remains in an idle state (power-down mode) until it is interrupted.

The idle state is exited after an unmasked interrupt, even if INTM = 1. If INTM = 1, the program continues executing at the instruction following the idle. If INTM = 0, the program branches to the corresponding interrupt service routine. The interrupt is enabled by the interrupt mask register (IMR), regardless of the INTM value. The following options, indicated by the value of K, determine the type of interrupts that can release the device from idle:

- K = 1      Peripherals, such as the timer and the serial ports, are still active. The peripheral interrupts as well as reset and external interrupts release the processor from idle mode.
- K = 2      Peripherals, such as the timer and the serial ports, are inactive. Reset and external interrupts release the processor from idle mode. Because interrupts are not latched in idle mode as they are in normal device operation, they must be low for a number of cycles to be acknowledged.
- K = 3      Peripherals, such as the timer and the serial ports, are inactive and the PLL is halted. Reset and external interrupts release the processor from idle mode. Because interrupts are not latched in idle mode as they are in normal device operation, they must be low for a number of cycles to be acknowledged.

**Note:**

This instruction is not repeatable.

## **IDLE** *Idle Until Interrupt*

---

<b>Words</b>	1 word
<b>Cycles</b>	The number of cycles needed to execute this instruction depends on the idle period. Because the entire device is halted when $K = 3$ , the number of cycles cannot be specified. The minimum number of cycles is 4.
<b>Classes</b>	Class 36 (see page 3-72)
<b>Example 1</b>	<p>IDLE 1</p> <p>The processor idles until a reset or unmasked interrupt occurs.</p>
<b>Example 2</b>	<p>IDLE 2</p> <p>The processor idles until a reset or unmasked external interrupt occurs.</p>
<b>Example 3</b>	<p>IDLE 3</p> <p>The processor idles until a reset or unmasked external interrupt occurs.</p>



**Syntax** **INTR K****Operands**  $0 \leq K \leq 31$ 

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	1	1	1	0	K	K	K	K	K

**Execution**

(SP) - 1 → SP  
(PC) + 1 → TOS  
interrupt vector specified by K → PC  
1 → INTM

**Status Bits** Affects INTM and IFR

**Description**

This instruction transfers program control to the interrupt vector specified by *K*. This instruction allows you to use your application software to execute any interrupt service routine. For a list of interrupts and their corresponding *K* value, see Appendix B.

During execution of the instruction, the PC is incremented by 1 and pushed onto the TOS. Then, the interrupt vector specified by *K* is loaded in the PC and the interrupt service routine for this interrupt is executed. The corresponding bit in the interrupt flag register (IFR) is cleared and interrupts are globally disabled (INTM = 1). The interrupt mask register (IMR) has no effect on the INTR instruction. INTR is executed regardless of the value of INTM.

**Note:**

This instruction is not repeatable.

**Words** 1 word**Cycles** 3 cycles**Classes** Class 35 (see page 3-72)**Example** INTR 3

	Before Instruction	After Instruction
PC	0025	FF8C
INTM	0	1
IPRT	01FF	01FF
SP	1000	0FFF
Data Memory		
0FFFh	9653	0026

## LD Load Accumulator With Shift

### Syntax

- 1: LD *Smem*, *dst*
- 2: LD *Smem*, **TS**, *dst*
- 3: LD *Smem*, **16**, *dst*
- 4: LD *Smem* [, *SHIFT*], *dst*
- 5: LD *Xmem*, *SHFT*, *dst*
- 6: LD **#K**, *dst*
- 7: LD **#lk** [, *SHFT*], *dst*
- 8: LD **#lk**, **16**, *dst*
- 9: LD *src*, **ASM** [, *dst*]
- 10: LD *src* [, *SHIFT*] [, *dst*]

For additional load instructions, see *Load T/DP/ASM/ARP* on page 4-70.

### Operands

*Smem*: Single data-memory operand  
*Xmem*: Dual data-memory operands  
*src*, *dst*: A (accumulator A)  
                   B (accumulator B)

$$0 \leq K \leq 255$$

$$-32\,768 \leq lk \leq 32\,767$$

$$-16 \leq SHIFT \leq 15$$

$$0 \leq SHFT \leq 15$$

### Opcode

- 1:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	D	I	A	A	A	A	A	A	A
- 2:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	D	I	A	A	A	A	A	A	A
- 3:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	D	I	A	A	A	A	A	A	A
- 4:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	0	D	0	1	0	S	H	I	F	T
- 5:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	D	X	X	X	X	S	H	F	T
- 6:
 

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	0	0	D	K	K	K	K	K	K	K	K

7:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	0	1	0	S	H	F	T
16-bit constant															

8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	1	1	0	0	0	1	0
16-bit constant															

9:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	0	0	0	1	0

10:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	1	0	S	H	I	F	T

**Execution**

- 1: (Smem) → dst
- 2: (Smem) << TS → dst
- 3: (Smem) << 16 → dst
- 4: (Smem) << SHIFT → dst
- 5: (Xmem) << SHFT → dst
- 6: K → dst
- 7: lk << SHFT → dst
- 8: lk << 16 → dst
- 9: (src) << ASM → dst
- 10: (src) << SHIFT → dst

**Status Bits**

Affected by SXM in all accumulator loads  
 Affected by OVM in loads with SHIFT or ASM shift  
 Affects OVdst (or OVsrc, when dst = src) in loads with SHIFT or ASM shift

**Description**

This instruction loads the accumulator (*dst*, or *src* if *dst* is not specified) with a data-memory value or an immediate value, supporting different shift quantities. Additionally, the instruction supports accumulator-to-accumulator moves with shift.

## LD Load Accumulator With Shift

### Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- ☐ Syntax 4: If  $SHIFT = 0$ , the instruction opcode is assembled as syntax 1.
- ☐ Syntax 4: If  $0 < SHIFT \leq 15$  and Smem indirect addressing mode is included in Xmem, the instruction opcode is assembled as syntax 5.
- ☐ Syntax 5: If  $SHIFT = 0$ , the instruction opcode is assembled as syntax 1.
- ☐ Syntax 7: If  $SHIFT = 0$  and  $0 \leq lk \leq 255$ , the instruction opcode is assembled as syntax 6.

### Words

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 word  
Syntaxes 4, 7, and 8: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

### Cycles

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 cycle  
Syntaxes 4, 7, and 8: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

### Classes

Syntaxes 1, 2, 3, and 5: Class 3A (see page 3-5)  
Syntaxes 1, 2, and 3: Class 3B (see page 3-6)  
Syntax 4: Class 4A (see page 3-7)  
Syntax 4: Class 4B (see page 3-8)  
Syntaxes 6, 9, and 10: Class 1 (see page 3-3)  
Syntaxes 7 and 8: Class 2 (see page 3-4)

### Example 1

LD \*AR1, A

Before Instruction		After Instruction	
A	00 0000 0000	A	00 0000 FEDC
SXM	0	SXM	0
AR1	0200	AR1	0200
Data Memory			
0200h	FEDC	0200h	FEDC

**Example 2**

LD \*AR1, A

	Before Instruction	After Instruction
A	00 0000 0000	FF FFFF FEDC
SXM	1	1
AR1	0200	0200
Data Memory		
0200h	FEDC	FEDC

**Example 3**

LD \*AR1, TS, B

	Before Instruction	After Instruction
B	00 0000 0000	FF FFFE DC00
SXM	1	1
AR1	0200	0200
T	8	8
Data Memory		
0200h	FEDC	FEDC

**Example 4**

LD \*AR3+, 16, A

	Before Instruction	After Instruction
A	00 0000 0000	FF FEDC 0000
SXM	1	1
AR3	0300	0301
Data Memory		
0300h	FEDC	FEDC

**Example 5**

LD #248, B

	Before Instruction	After Instruction
B	00 0000 0000	00 0000 00F8
SXM	1	1

**Example 6**

LD A, 8, B

	Before Instruction	After Instruction
A	00 7FFD 0040	00 7FF0 0040
B	00 0000 FFFF	7F FD00 4000
OVB	0	1
SXM	1	1
Data Memory		
0200h	FEDC	FEDC

## LD Load T/DP/ASM/ARP

### Syntax

- 1: LD Smem, T
- 2: LD Smem, DP
- 3: LD #k9, DP
- 4: LD #k5, ASM
- 5: LD #k3, ARP
- 6: LD Smem, ASM

For additional load instructions, see *Load Accumulator With Shift* on page 4-66.

### Operands

Smem: Single data-memory operand  
 $0 \leq k9 \leq 511$   
 $-16 \leq k5 \leq 15$   
 $0 \leq k3 \leq 7$

### Opcode

- 1:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 0  | 1  | 1  | 0  | 0  | 0 | 0 | I | A | A | A | A | A | A | A |
- 2:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 1  | 0  | 0  | 0  | 1  | 1 | 0 | I | A | A | A | A | A | A | A |
- 3:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 1  | 1  | 0  | 1  | 0  | 1 | K | K | K | K | K | K | K | K | K |
- 4:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 1  | 1  | 0  | 1  | 1  | 0 | 1 | 0 | 0 | 0 | K | K | K | K | K |
- 5:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 1  | 1  | 1  | 0  | 1  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | K | K | K |
- 6:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 0  | 1  | 1  | 0  | 0  | 1 | 0 | I | A | A | A | A | A | A | A |

### Execution

- 1: (Smem) → T
- 2: (Smem(8-0)) → DP
- 3: k9 → DP
- 4: k5 → ASM
- 5: k3 → ARP
- 6: (Smem(4-0)) → ASM

### Status Bits

None

**Description** This instruction loads a value into T or into the DP, ASM, and ARP fields of ST0 or ST1. The value loaded can be a single data-memory operand *Smem* or a constant.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** Syntaxes 1, 3, 4, 5, and 6: 1 cycle  
Syntax 2: 3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Syntaxes 1 and 6: Class 3A (see page 3-5)  
Syntaxes 1 and 6: Class 3B (see page 3-6)  
Syntax 2: Class 5A (see page 3-9)  
Syntax 2: Class 5B (see page 3-9)  
Syntaxes 3, 4, and 5: Class 1 (see page 3-3)

**Example 1**

LD \*AR3+, T

	Before Instruction	After Instruction
T	0000	FEDC
AR3	0300	0301
Data Memory		
0300h	FEDC	FEDC

**Example 2**

LD \*AR4, DP

	Before Instruction	After Instruction
AR4	0200	0200
DP	1FF	0DC
Data Memory		
0200h	FEDC	FEDC

**Example 3**

LD #23, DP

	Before Instruction	After Instruction
DP	1FF	017

**Example 4**

LD 15, ASM

	Before Instruction	After Instruction
ASM	00	0F

**Example 5**

LD 3, ARP

	Before Instruction	After Instruction
ARP	0	3

## **LD** *Load T/DP/ASM/ARP*

---

### **Example 6**

LD DAT0, ASM

	Before Instruction	After Instruction
ASM	<div>00</div>	<div>1C</div>
DP	<div>004</div>	<div>004</div>
Data Memory		
0200h	<div>FEDC</div>	<div>FEDC</div>



**Syntax** **LDM** *MMR*, *dst*

**Operands**      **MMR:**    Memory-mapped register  
                  **dst:**     A (accumulator)  
                             B (accumulator)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	D	I	A	A	A	A	A	A	A

**Execution** (MMR) → dst**Status Bits** None

**Description**      This instruction loads *dst* with the value memory memory-mapped register *MMR*. The nine MSBs of the effective address are cleared to 0 to designate data page 0, regardless of the current value of DP or the upper nine bits of ARx. This instruction is not affected by the value of SXM.

**Words** 1 word**Cycles** 1 cycle**Classes** Class 3A (see page 3-5)**Example 1** LDM AR4, A

	Before Instruction	After Instruction
A	00 0000 1111	00 0000 FFFF
AR4	FFFF	FFFF

**Example 2** LDM 060h, B

	Before Instruction	After Instruction
B	00 0000 0000	00 0000 1234
Data Memory		
0060h	1234	1234

## LD||MAC[R] Load Accumulator With Parallel Multiply Accumulate With/Without Rounding

Syntax	LD <i>Xmem</i> , <i>dst</i>    MAC[R] <i>Ymem</i> [, <i>dst_</i> ]																																
Operands	<i>dst</i> : A (accumulator A) B (accumulator B) <i>dst_</i> : If <i>dst</i> = A, then <i>dst_</i> = B; if <i>dst</i> = B, then <i>dst_</i> = A <i>Xmem</i> , <i>Ymem</i> : Dual data-memory operands																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>R</td><td>D</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	0	1	0	R	D	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	0	1	0	1	0	R	D	X	X	X	X	Y	Y	Y	Y																		
Execution	( <i>Xmem</i> ) << 16 → <i>dst</i> (31–16) If (Rounding) Round ((( <i>Ymem</i> ) × ( <i>T</i> )) + ( <i>dst_</i> )) → <i>dst_</i> Else (( <i>Ymem</i> ) × ( <i>T</i> )) + ( <i>dst_</i> ) → <i>dst_</i>																																
Status Bits	Affected by SXM, FRCT, and OVM Affects OV <i>dst_</i>																																
Description	<p>This instruction loads the high part of <i>dst</i> (bits 31–16) with a 16-bit dual data-memory operand <i>Xmem</i> shifted left 16-bits. In parallel, this instruction multiplies a dual data-memory operand <i>Ymem</i> by the content of <i>T</i>, adds the result of the multiplication to <i>dst_</i>, and stores the result in <i>dst_</i>.</p> <p>If you use the R suffix, this instruction optionally rounds the result of the multiply and accumulate operation by adding 2<sup>15</sup> to the result and clearing the LSBs (15–0) to 0, and stores the result in <i>dst_</i>.</p>																																
Words	1 word																																
Cycles	1 cycle																																
Classes	Class 7 (see page 3-12)																																
Example 1	LD *AR4+, A																																

	Before Instruction	After Instruction
A	00 0000 1000	00 1234 0000
B	00 0000 1111	00 010C 9511
T	0400	0400
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
0100h	1234	1234
0200h	4321	4321

**Example 2**

LD \*AR4+, A  
||MACR \*AR5+, B

	Before Instruction	After Instruction
A	00 0000 1000	00 1234 0000
B	00 0000 1111	00 010D 0000
T	0400	0400
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
0100h	1234	1234
0200h	4321	4321

**LD||MAS[R]** *Load Accumulator With Parallel Multiply Subtract With/Without Rounding*

**Syntax**                **LD** *Xmem*, *dst*  
                         **|| MAS[R]** *Ymem* [, *dst\_* ]

**Operands**            *Xmem*, *Ymem*:    Dual data-memory operands  
                         *dst*:                A (accumulator A)  
                                                B (accumulator B)  
                         *dst\_*:              If *dst* = A, then *dst\_* = B; if *dst* = B, then *dst\_* = A

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	R	D	X	X	X	X	Y	Y	Y	Y

**Execution**            (*Xmem*) << 16 → *dst* (31–16)  
                         If (Rounding)  
                             Round ((*dst\_*) – ((*T*) × (*Ymem*))) → *dst\_*  
                         Else  
                             (*dst\_*) – ((*T*) × (*Ymem*)) → *dst\_*

**Status Bits**           Affected by SXM, FRCT, and OVM  
                         Affects OV*dst\_*

**Description**           This instruction loads the high part of *dst* (bits 31–16) with a 16-bit dual data-memory operand *Xmem* shifted left 16 bits. In parallel, this instruction multiplies a dual data-memory operand *Ymem* by the content of *T*, subtracts the result of the multiplication from *dst\_*, and stores the result in *dst\_*.

If you use the R suffix, this instruction optionally rounds the result of the multiply and subtract operation by adding  $2^{15}$  to the result and clearing the LSBs (15–0) to 0, and stores the result in *dst\_*.

**Words**                1 word

**Cycles**               1 cycle

**Classes**              Class 7 (see page 3-12)

**Example 1**

LD \*AR4+, A  
||MAS \*AR5+, B

Before Instruction		After Instruction	
A	00 0000 1000	A	00 1234 0000
B	00 0000 1111	B	FF FEF3 8D11
T	0400	T	0400
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	1234	0100h	1234
0200h	4321	0200h	4321

**Example 2**

LD \*AR4+, A  
||MASR \*AR5+, B

Before Instruction		After Instruction	
A	00 0000 1000	A	00 1234 0000
B	00 0000 1111	B	FF FEF4 0000
T	0400	T	0400
FRCT	0	FRCT	0
AR4	0100	AR4	0101
AR5	0200	AR5	0201
Data Memory			
0100h	1234	0100h	1234
0200h	4321	0200h	4321

## LDR Load Memory Value in Accumulator High With Rounding

**Syntax** LDR *Smem*, *dst*

**Operands** *Smem*: Single data-memory operand  
*dst*: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	1	D	I	A	A	A	A	A	A	A

**Execution** (*Smem*) << 16 + 1 << 15 → *dst*(31–16)

**Status Bits** Affected by SXM

**Description** This instruction loads the data-memory value *Smem* shifted left 16 bits into the high part of *dst* (bits 31–16). *Smem* is rounded by adding  $2^{15}$  to this value and clearing the LSBs (15–0) of the accumulator to 0. Bit 15 of the accumulator is set to 1.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 3A (see page 3-5)  
Class 3B (see page 3-6)

**Example** LDR \*AR1, A

Before Instruction		After Instruction	
A	00 0000 0000	A	00 FEDC 8000
SXM	0	SXM	0
AR1	0200	AR1	0200
Data Memory			
0200h	FEDC	0200h	FEDC

**Syntax** **LDU** *Smem, dst*

**Operands**      *Smem*: Single data-memory operand  
                      *dst*:    A (accumulator A)  
                                  B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	D	I	A	A	A	A	A	A	A

**Execution**      (*Smem*) → *dst*(15–0)  
                      00 0000h → *dst*(39–16)

**Status Bits**      None

**Description**      This instruction loads the data-memory value *Smem* into the low part of *dst* (bits 15–0). The guard bits and the high part of *dst* (bits 39–16) are cleared to 0. Data is then treated as an unsigned 16-bit number. There is no sign extension regardless of the status of the SXM bit.

**Words**            1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles**            1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes**            Class 3A (see page 3-5)  
                           Class 3B (see page 3-6)

**Example**            LDU \*AR1, A

	Before Instruction	After Instruction
A	00 0000 0000	00 0000 FEDC
AR1	0200	0200
Data Memory		
0200h	FEDC	FEDC

## LMS *Least Mean Square*

**Syntax** LMS *Xmem, Ymem*

**Operands** Xmem, Ymem: Dual data-memory operands

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	1	X	X	X	X	Y	Y	Y	Y

**Execution**

$$((A) + (Xmem)) \ll 16 + 2^{15} \rightarrow A$$

$$(B) + (Xmem) \times (Ymem) \rightarrow B$$

**Status Bits** Affected by SXM, FRCT, and OVM  
Affects C, OVA, and OVB

**Description** This instruction executes the least mean square (LMS) algorithm. The dual data-memory operand *Xmem* is added to accumulator A and shifted left 16 bits. The result is rounded by adding  $2^{15}$  to the high part of the accumulator (bits 31–16). The result is stored in accumulator A. In parallel, *Xmem* and *Ymem* are multiplied and the result is added to accumulator B. *Xmem* does not overwrite T; therefore, T always contains the error value used to update coefficients.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 7 (see page 3-12)

**Example** LMS \*AR3+, \*AR4+

	Before Instruction	After Instruction
A	00 7777 8888	00 77CD 0888
B	00 0000 0100	00 0000 3972
FRCT	0	0
AR3	0100	0101
AR4	0200	0201
Data Memory		
0100h	0055	0055
0200h	00AA	00AA



**Syntax** LTD *Smem***Operands** Smem: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	I	A	A	A	A	A	A	A

**Execution**

(Smem) → T  
 (Smem) → Smem + 1

**Status Bits** None

**Description**

This instruction copies the content of a single data-memory location *Smem* into T and into the address following this data-memory location. When data is copied, the content of the address location remains the same. This function is useful for implementing a Z delay in digital signal processing applications. This function also contains the memory delay instruction (page 4-41).

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 24A (see page 3-56)

Class 24B (see page 3-56)

**Example** LTD \*AR3

	Before Instruction	After Instruction
T	0000	6CAC
AR3	0100	0100
Data Memory		
0100h	6CAC	6CAC
0101h	xxxx	6CAC

## MAC[R] *Multiply Accumulate With/Without Rounding*

### Syntax

- 1: **MAC[R]** *Smem, src*
- 2: **MAC[R]** *Xmem, Ymem, src [, dst]*
- 3: **MAC** *#lk, src [, dst]*
- 4: **MAC** *Smem, #lk, src [, dst]*

### Operands

*Smem*: Single data-memory operands  
*Xmem, Ymem*: Dual data-memory operands  
*src, dst*: A (accumulator A)  
              B (accumulator B)  
 $-32\,768 \leq lk \leq 32\,767$

### Opcode

- 1:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 0  | 1  | 0  | 1  | 0  | R | S | I | A | A | A | A | A | A | A |
- 2:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 0  | 1  | 1  | 0  | R  | S | D | X | X | X | X | Y | Y | Y | Y |
- 3:
- |                 |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|-----------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15              | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1               | 1  | 1  | 1  | 0  | 0  | S | D | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 16-bit constant |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
- 4:
- |                 |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|-----------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15              | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0               | 1  | 1  | 0  | 0  | 1  | S | D | I | A | A | A | A | A | A | A |
| 16-bit constant |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |

### Execution

- 1:  $(Smem) \times (T) + (src) \rightarrow src$
- 2:  $(Xmem) \times (Ymem) + (src) \rightarrow dst$   
 $(Xmem) \rightarrow T$
- 3:  $(T) \times lk + (src) \rightarrow dst$
- 4:  $(Smem) \times lk + (src) \rightarrow dst$   
 $(Smem) \rightarrow T$

### Status Bits

Affected by FRCT and OVM  
Affects OVdst (or OVsrc, if dst is not specified)

### Description

This instruction multiplies and adds with or without rounding. The result is stored in *dst* or *src*, as specified. For syntaxes 2 and 4, the data-memory value after the instruction is stored in T. T is updated in the read phase.

If you use the R suffix, this instruction rounds the result of the multiply and accumulate operation by adding  $2^{15}$  to the result and clearing the LSBs (15–0) to 0.

**Words**

Syntaxes 1 and 2: 1 word  
Syntaxes 3 and 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles**

Syntaxes 1 and 2: 1 cycle  
Syntaxes 3 and 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes**

Syntax 1: Class 3A (see page 3-5)  
Syntax 1: Class 3B (see page 3-6)  
Syntax 2: Class 7 (see page 3-12)  
Syntax 3: Class 2 (see page 3-4)  
Syntax 4: Class 6A (see page 3-10)  
Syntax 4: Class 6B (see page 3-11)

**Example 1**

MAC \*AR5+, A

	Before Instruction	After Instruction
A	00 0000 1000	00 0048 E000
T	0400	0400
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	1234	1234

**Example 2**

MAC #345h, A, B

	Before Instruction	After Instruction
A	00 0000 1000	00 0000 1000
B	00 0000 0000	00 001A 3800
T	0400	0400
FRCT	1	1

**Example 3**

MAC \*AR5+, #1234h, A

	Before Instruction	After Instruction
A	00 0000 1000	00 0626 1060
T	0000	5678
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	5678	5678

## MAC[R] *Multiply Accumulate With/Without Rounding*

### Example 4

MAC \*AR5+, \*AR6+, A, B

Before Instruction		After Instruction	
A	00 0000 1000	A	00 0000 1000
B	00 0000 0004	B	00 0C4C 10C0
T	0008	T	5678
FRCT	1	FRCT	1
AR5	0100	AR5	0101
AR6	0200	AR6	0201
Data Memory			
0100h	5678	0100h	5678
0200h	1234	0200h	1234

### Example 5

MACR \*AR5+, A

Before Instruction		After Instruction	
A	00 0000 1000	A	00 0049 0000
T	0400	T	0400
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	1234	0100h	1234

### Example 6

MACR \*AR5+, \*AR6+, A, B

Before Instruction		After Instruction	
A	00 0000 1000	A	00 0000 1000
B	00 0000 0004	B	00 0C4C 0000
T	0008	T	5678
FRCT	1	FRCT	1
AR5	0100	AR5	0101
AR6	0200	AR6	0201
Data Memory			
0100h	5678	0100h	5678
0200h	1234	0200h	1234

## Multiply by Accumulator A and Accumulate With/Without Rounding **MACA[R]**

Syntax	1: <b>MACA[R]</b> <i>Smem</i> [, <i>B</i> ] 2: <b>MACA[R]</b> <i>T</i> , <i>src</i> [, <i>dst</i> ]																																																																
Operands	Smem: Single data-memory operand src, dst: A (accumulator A) B (accumulator B)																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>R</td><td>1</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>S</td><td>D</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>R</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	1	0	1	R	1	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	S	D	1	0	0	0	1	0	0	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
0	0	1	1	0	1	R	1	I	A	A	A	A	A	A	A																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
1	1	1	1	0	1	S	D	1	0	0	0	1	0	0	R																																																		
Execution	1: (Smem) × (A(32–16)) + (B) → B (Smem) → T 2: (T) × (A(32–16)) + (src) → src																																																																
Status Bits	Affected by FRCT and OVM Affects OVdst (or OVsrc, if dst is not specified) and OVB in syntax 1																																																																
Description	<p>This instruction multiplies the high part of accumulator A (bits 32–16) by a single data-memory operand <i>Smem</i> or by the content of T, adds the product to accumulator B (syntax 1) or to <i>src</i>. The result is stored in accumulator B (syntax 1) or in <i>dst</i> or <i>src</i> if no <i>dst</i> is specified. A(32–16) is used as a 17-bit operand for the multiplier.</p> <p>If you use the R suffix, this instruction rounds the result of the multiply by accumulator A operation by adding 2<sup>15</sup> to the result and clearing the 16 LSBs of <i>dst</i> (bits 15–0) to 0.</p>																																																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.																																																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.																																																																
Classes	Syntaxes 1 and 2: Class 3A (see page 3-5) Syntaxes 1 and 2: Class 3B (see page 3-6) Syntaxes 3 and 4: Class 1 (see page 3-3)																																																																

## MACA[R] *Multiply by Accumulator A and Accumulate With/Without Rounding*

### Example 1

MACA \*AR5+

Before Instruction		After Instruction	
A	00 1234 0000	A	00 1234 0000
B	00 0000 0000	B	00 0626 0060
T	0400	T	5678
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	5678	0100h	5678

### Example 2

MACA T, B, B

Before Instruction		After Instruction	
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	00 009D 4BA0
T	0444	T	0444
FRCT	1	FRCT	1

### Example 3

MACAR \*AR5+, B

Before Instruction		After Instruction	
A	00 1234 0000	A	00 1234 0000
B	00 0000 0000	B	00 0626 0000
T	0400	T	5678
FRCT	0	FRCT	0
AR5	0100	AR5	0101
Data Memory			
0100h	5678	0100h	5678

### Example 4

MACAR T, B, B

Before Instruction		After Instruction	
A	00 1234 0000	A	00 1234 0000
B	00 0002 0000	B	00 009D 0000
T	0444	T	0444
FRCT	1	FRCT	1

Assembly Language Instructions 4-87

## MACD *Multiply by Program Memory and Accumulate With Delay*

### Example

MACD \*AR3-, COEFS, A

	Before Instruction	After Instruction
A	00 0077 0000	00 007D 0B44
T	0008	0055
FRCT	0	0
AR3	0100	00FF
Program Memory		
COEFS	1234	1234
Data Memory		
0100h	0055	0055
0101h	0066	0055



**Syntax** **MACP** *Smem, pmad, src*

**Operands** *Smem*: Single data-memory operand  
*src*: A (accumulator A)  
           B (accumulator B)  
 $0 \leq pmad \leq 65535$

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	1	1	0	0	S	I	A	A	A	A	A	A	A
	16-bit constant															

**Execution** (*pmad*) → PAR  
 If (RC) ≠ 0  
 Then  
     (*Smem*) × (Pmem addressed by PAR) + (*src*) → *src*  
     (*Smem*) → T  
     (PAR) + 1 → PAR  
 Else  
     (*Smem*) × (Pmem addressed by PAR) + (*src*) → *src*  
     (*Smem*) → T

**Status Bits** Affected by FRCT and OVM  
 Affects OVsrc

**Description** This instruction multiplies a single data-memory value *Smem* by a program-memory value *pmad*, adds the product to *src*, and stores the result in *src*. The data-memory value *Smem* is copied into T. When this instruction is repeated, the program-memory address (in the program address register PAR) is incremented by 1. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 22A (see page 3-50)  
 Class 22B (see page 3-52)

## MACP *Multiply by Program Memory and Accumulate*

### Example

MACP \*AR3-, COEFS, A

	Before Instruction	After Instruction
A	00 0077 0000	00 007D 0B44
T	0008	0055
FRCT	0	0
AR3	0100	00FF
Program Memory		
COEFS	1234	1234
Data Memory		
0100h	0055	0055
0101h	0066	0066

**Syntax** **MACSU** *Xmem*, *Ymem*, *src*

**Operands** *Xmem*, *Ymem*: Dual data-memory operands  
*src*: A (accumulator A)  
 B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	S	X	X	X	X	Y	Y	Y	Y

**Execution**  $\text{unsigned}(Xmem) \times \text{signed}(Ymem) + (src) \rightarrow src$   
 $(Xmem) \rightarrow T$

**Status Bits** Affected by FRCT and OVM  
 Affects OVsrc

**Description** This instruction multiplies an unsigned data-memory value *Xmem* by a signed data-memory value *Ymem*, adds the product to *src*, and stores the result in *src*. The 16-bit unsigned value *Xmem* is stored in T. T is updated with the unsigned value *Xmem* in the read phase.

The data addressed by *Xmem* is fed from the D bus. The data addressed by *Ymem* is fed from the C bus.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 7 (see page 3-12)

**Example** MACSU \*AR4+, \*AR5+, A

	Before Instruction	After Instruction
A	00 0000 1000	00 09A0 AA84
T	0008	8765
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
0100h	8765	8765
0200h	1234	1234

## MAR *Modify Auxiliary Register*

**Syntax** MAR *Smem*

**Operands** Smem: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	0	1	I	A	A	A	A	A	A	A

**Execution** In indirect addressing mode, the auxiliary register is modified as follows:

If compatibility is on (CMPT = 1), then:

If (ARx = AR0)

AR(ARP) is modified

ARP is unchanged

Else

ARx is modified

x → ARP

Else compatibility is off (CMPT = 0)

ARx is modified

ARP is unchanged

**Status Bits** Affected by CMPT

Affects ARP (if CMPT = 1)

**Description** This instruction modifies the content of the selected auxiliary register (ARx) as specified by *Smem*. In compatibility mode (CMPT = 1), this instruction modifies the ARx content as well as the auxiliary register pointer (ARP) value.

If CMPT = 0, the auxiliary register is modified but ARP is not.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 1 (see page 3-3)

Class 2 (see page 3-4)

**Example 1** MAR \*AR3+

	Before Instruction	After Instruction
CMPT	<input type="text" value="0"/>	<input type="text" value="0"/>
ARP	<input type="text" value="0"/>	<input type="text" value="0"/>
AR3	<input type="text" value="0100"/>	<input type="text" value="0101"/>

**Example 2**

MAR \*-

	Before Instruction	After Instruction
CMPT	<input type="text" value="1"/>	<input type="text" value="1"/>
ARP	<input type="text" value="4"/>	<input type="text" value="4"/>
AR4	<input type="text" value="0100"/>	<input type="text" value="00FF"/>

**Example 3**

MAR \*AR3

	Before Instruction	After Instruction
CMPT	<input type="text" value="1"/>	<input type="text" value="1"/>
ARP	<input type="text" value="0"/>	<input type="text" value="3"/>
AR0	<input type="text" value="0008"/>	<input type="text" value="0008"/>
AR3	<input type="text" value="0100"/>	<input type="text" value="00F8"/>

**Example 4**

MAR \*+AR3

	Before Instruction	After Instruction
CMPT	<input type="text" value="1"/>	<input type="text" value="1"/>
ARP	<input type="text" value="0"/>	<input type="text" value="3"/>
AR3	<input type="text" value="0100"/>	<input type="text" value="0164"/>

**Example 5**

MAR \*+AR3

	Before Instruction	After Instruction
CMPT	<input type="text" value="1"/>	<input type="text" value="1"/>
ARP	<input type="text" value="0"/>	<input type="text" value="3"/>
AR3	<input type="text" value="0100"/>	<input type="text" value="009C"/>

## MAS[R] *Multiply and Subtract With/Without Rounding*

<b>Syntax</b>	1: <b>MAS[R]</b> <i>Smem</i> , <i>src</i> 2: <b>MAS[R]</b> <i>Xmem</i> , <i>Ymem</i> , <i>src</i> [, <i>dst</i> ]																																																																
<b>Operands</b>	<i>Smem</i> : Single data-memory operand <i>Xmem</i> , <i>Ymem</i> : Dual data-memory operands <i>src</i> , <i>dst</i> : A (accumulator A) B (accumulator B)																																																																
<b>Opcode</b>	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>R</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>R</td><td>S</td><td>D</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	0	1	1	R	S	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	1	1	1	R	S	D	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
0	0	1	0	1	1	R	S	I	A	A	A	A	A	A	A																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
1	0	1	1	1	R	S	D	X	X	X	X	Y	Y	Y	Y																																																		
<b>Execution</b>	1: ( <i>src</i> ) – ( <i>Smem</i> ) × ( <i>T</i> ) → <i>src</i> 2: ( <i>src</i> ) – ( <i>Xmem</i> ) × ( <i>Ymem</i> ) → <i>src</i> ( <i>Xmem</i> ) → <i>T</i>																																																																
<b>Status Bits</b>	Affected by FRCT and OVM Affects OVdst (or OVsrc, if <i>dst</i> = <i>src</i> )																																																																
<b>Description</b>	<p>This instruction multiplies an operand by the content of <i>T</i> or multiplies two operands, subtracts the result from <i>src</i>, if specified, or <i>dst</i>, and stores the result in <i>src</i> or <i>dst</i>, as specified. <i>Xmem</i> is loaded into <i>T</i> in the read phase.</p> <p>If you use the R suffix, this instruction rounds the result of the multiply and subtract operation by adding 2<sup>15</sup> to the result and clearing bits 15–0 of the result to 0.</p> <p>The data addressed by <i>Xmem</i> is fed from DB and the data addressed by <i>Ymem</i> is fed from CB.</p>																																																																
<b>Words</b>	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																																																
<b>Cycles</b>	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																																																
<b>Classes</b>	Syntax 1: Class 3A (see page 3-5) Syntax 1: Class 3B (see page 3-6) Syntax 2: Class 7 (see page 3-12)																																																																

### Example 1

MAS \*AR5+, A

	Before Instruction	After Instruction
A	00 0000 1000	FF FFB7 4000
T	0400	0400
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	1234	1234

### Example 2

MAS \*AR5+, \*AR6+, A, B

	Before Instruction	After Instruction
A	00 0000 1000	00 0000 1000
B	00 0000 0004	FF F3B4 0F40
T	0008	5678
FRCT	1	1
AR5	0100	0101
AR6	0200	0201
Data Memory		
0100h	5678	5678
0200h	1234	1234

### Example 3

MASR \*AR5+, A

	Before Instruction	After Instruction
A	00 0000 1000	FF FFB7 0000
T	0400	0400
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	1234	1234

## MAS[R] *Multiply and Subtract With/Without Rounding*

### Example 4

MASR \*AR5+, \*AR6+, A, B

Before Instruction		After Instruction	
A	00 0000 1000	A	00 0000 1000
B	00 0000 0004	B	FF F3B4 0000
T	0008	T	5678
FRCT	1	FRCT	1
AR5	0100	AR5	0101
AR6	0200	AR6	0201
Data Memory			
0100h	5678	0100h	5678
0200h	1234	0200h	1234



Syntax	1: <b>MASA</b> <i>Smem</i> [, <i>B</i> ] 2: <b>MASA[R]</b> <i>T</i> , <i>src</i> [, <i>dst</i> ]																																																																
Operands	Smem: Single data-memory operand src, dst: A (accumulator A) B (accumulator B)																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>S</td><td>D</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>R</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	1	0	0	1	1	1	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	S	D	1	0	0	0	1	0	1	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
0	0	1	1	0	0	1	1	1	A	A	A	A	A	A	A																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
1	1	1	1	0	1	S	D	1	0	0	0	1	0	1	R																																																		
Execution	1: (B) – (Smem) × (A(32–16)) → B (Smem) → T 2: (src) – (T) × (A(32–16)) → dst																																																																
Status Bits	Affected by FRCT and OVM Affects OVdst (or OVsrc, if dst is not specified) and OVB in syntax 1																																																																
Description	<p>This instruction multiplies the high part of accumulator A (bits 32–16) by a single data-memory operand <i>Smem</i> or by the content of T, subtracts the result from accumulator B (syntax 1) or from <i>src</i>. The result is stored in accumulator B (syntax 1) or in <i>dst</i> or <i>src</i>, if no <i>dst</i> is specified. T is updated with the <i>Smem</i> value in the read phase.</p> <p>If you use the R suffix in syntax 2, this instruction optionally rounds the result of the multiply by accumulator A and subtract operation by adding 2<sup>15</sup> to the result and clearing bits 15–0 of the result to 0.</p>																																																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.																																																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.																																																																
Classes	Syntax 1: Class 3A (see page 3-5) Syntax 1: Class 3B (see page 3-6) Syntax 2: Class 1 (see page 3-3)																																																																

## MASA[R] *Multiply by Accumulator A and Subtract With/Without Rounding*

### Example 1

MASA \*AR5+

	Before Instruction	After Instruction
A	00 1234 0000	00 1234 0000
B	00 0002 0000	FF F9DB FFA0
T	0400	5678
FRCT	0	0
AR5	0100	0101
Data Memory		
0100h	5678	5678

### Example 2

MASA T, B

	Before Instruction	After Instruction
A	00 1234 0000	00 1234 0000
B	00 0002 0000	FF FF66 B460
T	0444	0444
FRCT	1	1

### Example 3

MASAR T, B

	Before Instruction	After Instruction
A	00 1234 0000	00 1234 0000
B	00 0002 0000	FF FF67 0000
T	0444	0444
FRCT	1	1

**Syntax** **MAX** *dst***Operands** *dst*: A (accumulator A)  
B (accumulator B)**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	D	1	0	0	0	0	1	1	0

**Execution**  
If (A > B)  
Then  
    (A) → *dst*  
    0 → C  
Else  
    (B) → *dst*  
    1 → C**Status Bits** Affects C**Description** This instruction compares the content of the accumulators and stores the maximum value in *dst*. If the maximum value is in accumulator A, the carry bit, C, is cleared to 0; otherwise, it is set to 1.**Words** 1 word**Cycles** 1 cycle**Classes** Class 1 (see page 3-3)**Example 1** MAX A

Before Instruction			After Instruction		
A	FFF6	-10	A	FFF6	-10
B	FFCB	-53	B	FFCB	-53
C	1		C	0	

**Example 2** MAX A

Before Instruction			After Instruction		
A	00 0000 0055		A	00 0000 1234	
B	00 0000 1234		B	00 0000 1234	
C	0		C	1	

## MIN Accumulator Minimum

**Syntax** MIN *dst*

**Operands** dst: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	D	1	0	0	0	0	1	1	1

**Execution**

If (A < B)  
Then  
    (A) → dst  
    0 → C  
Else  
    (B) → dst  
    1 → C

**Status Bits** Affects C

**Description** This instruction compares the content of the accumulators and stores the minimum value in *dst*. If the minimum value is in accumulator A, the carry bit, C, is cleared to 0; otherwise, it is set to 1.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1** MIN A

Before Instruction			After Instruction		
A	FFCB	-53	A	FFCB	-53
B	FFF6	-10	B	FFF6	-10
C	1		C	0	

**Example 2** MIN A

Before Instruction			After Instruction		
A	00 0000 1234		A	00 0000 1234	
B	00 0000 1234		B	00 0000 1234	
C	0		C	1	

**Syntax**

- 1: **MPY[R]** *Smem*, *dst*
- 2: **MPY** *Xmem*, *Ymem*, *dst*
- 3: **MPY** *Smem*, *#lk*, *dst*
- 4: **MPY** *#lk*, *dst*

**Operands**

*Smem*: Single data-memory operand  
*Xmem*, *Ymem*: Dual data-memory operands  
*dst*: A (accumulator A)  
           B (accumulator B)  
 $-32\,768 \leq lk \leq 32\,767$

**Opcode**

1:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	0	0	0	R	D	I	A	A	A	A	A	A	A

2:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	0	0	1	0	D	X	X	X	X	Y	Y	Y	Y

3:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	0	0	0	1	D	I	A	A	A	A	A	A	A
	16-bit constant															

4:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	0	0	0	D	0	1	1	0	0	1	1	0
	16-bit constant															

**Execution**

- 1:  $(T) \times (Smem) \rightarrow dst$
- 2:  $(Xmem) \times (Ymem) \rightarrow dst$   
 $(Xmem) \rightarrow T$
- 3:  $(Smem) \times lk \rightarrow dst$   
 $(Smem) \rightarrow T$
- 4:  $(T) \times lk \rightarrow dst$

**Status Bits**

Affected by FRCT and OVM  
 Affects OVDst

**Description**

This instruction multiplies the content of T or a data-memory value by a data-memory value or an immediate value, and stores the result in *dst*. T is loaded with the *Smem* or *Xmem* value in the read phase.

If you use the R suffix, this instruction optionally rounds the result of the multiply operation by adding  $2^{15}$  to the result and then clearing bits 15–0 to 0.

## MPY[R] *Multiply With/Without Rounding*

<b>Words</b>	Syntaxes 1 and 2: 1 word
	Syntaxes 3 and 4: 2 words
	Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.
<b>Cycles</b>	Syntaxes 1 and 2: 1 cycle
	Syntaxes 3 and 4: 2 cycles
	Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.
<b>Classes</b>	Syntax 1: Class 3A (see page 3-5)
	Syntax 1: Class 3B (see page 3-6)
	Syntax 2: Class 7 (see page 3-12)
	Syntax 3: Class 6A (see page 3-10)
	Syntax 3: Class 6B (see page 3-11)
	Syntax 4: Class 2 (see page 3-4)

### Example 1

MPY DAT13, A

Before Instruction		After Instruction	
A	00 0000 0036	A	00 0000 0054
T	0006	T	0006
FRCT	1	FRCT	1
DP	008	DP	008
Data Memory			
040Dh	0007	040Dh	0007

### Example 2

MPY \*AR2-, \*AR4+0%, B;

Before Instruction		After Instruction	
B	FF FFFF FFE0	B	00 0000 0020
FRCT	0	FRCT	0
AR0	0001	AR0	0001
AR2	01FF	AR2	01FE
AR4	0300	AR4	0301
Data Memory			
01FFh	0010	01FFh	0010
0300h	0002	0300h	0002

### Example 3

MPY #0FFFEh, A

Before Instruction		After Instruction	
A	000 0000 1234	A	FF FFFF C000
T	2000	T	2000
FRCT	0	FRCT	0

**Example 4**

MPYR DAT0, B

	Before Instruction	After Instruction
B	FF FE00 0001	00 0626 0000
T	1234	1234
FRCT	0	0
DP	004	004
Data Memory		
0200h	5678	5678

## MPYA *Multiply by Accumulator A*

**Syntax** 1: **MPYA** *Smem*

2: **MPYA** *dst*

**Operands** *Smem*: Single data-memory operand

*dst*: A (accumulator A)

B (accumulator B)

**Opcode**

1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	1	1	A	A	A	A	A	A	A

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	D	1	0	0	0	1	1	0	0

**Execution** 1: (*Smem*) × (A(32–16)) → B

(*Smem*) → T

2: (T) × (A(32–16)) → *dst*

**Status Bits** Affected by FRCT and OVM

Affects OVdst (OVb in syntax 1)

**Description** This instruction multiplies the high part of accumulator A (bits 32–16) by a single data-memory operand *Smem* or by the content of T, and stores the result in *dst* or accumulator B. T is updated in the read phase.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Syntax 1: Class 3A (see page 3-5)

Syntax 1: Class 3B (see page 3-6)

Syntax 2: Class 1 (see page 3-3)

**Example 1**

MPYA \*AR2

	Before Instruction	After Instruction
A	FF 8765 1111	FF 8765 1111
B	00 0000 0320	FF D743 6558
T	1234	5678
FRCT	0	0
AR2	0200	0200
Data Memory		
0200h	5678	5678



**Example 2**

MPYA B

Before Instruction		After Instruction	
A	FF 8765 1111	A	FF 8765 1111
B	00 0000 0320	B	FF DF4D B2A3
T	4567	T	4567
FRCT	0	FRCT	0

## MPYU *Multiply Unsigned*

**Syntax** **MPYU** *Smem, dst*

**Operands** **Smem:** Single data-memory operand  
**dst:** A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1	0	D	I	A	A	A	A	A	A	A

**Execution** unsigned(T) × unsigned(Smem) → dst

**Status Bits** Affected by FRCT and OVM  
Affects OVdst

**Description** This instruction multiplies the unsigned content of T by the unsigned content of the single data-memory operand *Smem*, and stores the result in *dst*. The multiplier acts as a signed 17 × 17-bit multiplier for this instruction with the MSB of both operands cleared to 0. This instruction is particularly useful for computing multiple-precision products, such as multiplying two 32-bit numbers to yield a 64-bit product.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 3A (see page 3-5)  
Class 3B (see page 3-6)

**Example** MPYU \*AR0-, A

	Before Instruction	After Instruction
A	FF 8000 0000	00 3F80 0000
T	4000	4000
FRCT	0	0
AR0	1000	0FFF
Data Memory		
1000h	FE00	FE00

*Move Data From Data Memory to Data Memory With X, Y Addressing*    **MVDD**

---

Syntax	MVDD <i>Xmem</i> , <i>Ymem</i>																																
Operands	<i>Xmem</i> , <i>Ymem</i> : Dual data-memory operands																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	0	0	1	0	1	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	1	1	0	0	1	0	1	X	X	X	X	Y	Y	Y	Y																		
Execution	( <i>Xmem</i> ) → <i>Ymem</i>																																
Status Bits	None																																
Description	This instruction copies the content of the data-memory location addressed by <i>Xmem</i> to the data-memory location addressed by <i>Ymem</i> .																																
Words	1 word																																
Cycles	1 cycle																																
Classes	Class 14 (see page 3-30)																																
Example	MVDD *AR3+, *AR5+																																

	Before Instruction		After Instruction		
AR3	<table><tr><td>8000</td></tr></table>	8000	AR3	<table><tr><td>8001</td></tr></table>	8001
8000					
8001					
AR5	<table><tr><td>0200</td></tr></table>	0200	AR5	<table><tr><td>0201</td></tr></table>	0201
0200					
0201					
Data Memory					
0200h	<table><tr><td>ABCD</td></tr></table>	ABCD	0200h	<table><tr><td>1234</td></tr></table>	1234
ABCD					
1234					
8000h	<table><tr><td>1234</td></tr></table>	1234	8000h	<table><tr><td>1234</td></tr></table>	1234
1234					
1234					

## MVDK Move Data From Data Memory to Data Memory With Destination Addressing

**Syntax** MVDK *Smem*, *dmad*

**Operands** *Smem*: Single data-memory operand  
 $0 \leq dmad \leq 65\,535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** (*dmad*) → EAR  
If (RC) ≠ 0  
Then  
    (*Smem*) → Dmem addressed by EAR  
    (EAR) + 1 → EAR  
Else  
    (*Smem*) → Dmem addressed by EAR

**Status Bits** None

**Description** This instruction copies the content of a single data-memory operand *Smem* to a data-memory location addressed by a 16-bit immediate value *dmad* (address is in the EAB address register EAR). You can use this instruction with the single-repeat instruction to move consecutive words in data memory (using indirect addressing). The number of words to be moved is one greater than the number contained in the repeat counter at the beginning of the instruction. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 19A (see page 3-40)  
Class 19B (see page 3-42)

**Example 1** MVDK DAT10, 8000h

	Before Instruction	After Instruction
DP	004	004
Data Memory		
020Ah	1234	1234
8000h	ABCD	1234

*Move Data From Data Memory to Data Memory With Destination Addressing* **MVDK**

**Example 2**

MVDK \*AR3-, 1000h

	Before Instruction	After Instruction
AR3	01FF	01FE
Data Memory		
1000h	ABCD	1234
01FFh	1234	1234

## MVDM *Move Data From Data Memory to Memory-Mapped Register*

**Syntax** MVDM *dmad*, *MMR*

**Operands** MMR: Memory-mapped register  
 $0 \leq \text{dmad} \leq 65\,535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	0	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** *dmad* → DAR  
 If (RC) ≠ 0  
 Then  
   (Dmem addressed by DAR) → MMR  
   (DAR) + 1 → DAR  
 Else  
   (Dmem addressed by DAR) → MMR

**Status Bits** None

**Description** This instruction copies data from a data-memory location *dmad* (address is in the DAB address register DAR) to a memory-mapped register *MMR*. The data-memory value is addressed with a 16-bit immediate value. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

**Cycles** 2 cycles

**Classes** Class 19A (see page 3-40)

**Example** MVDM 300h, BK

	Before Instruction	After Instruction
BK	<div style="border: 1px solid black; padding: 2px; display: inline-block;">ABCD</div>	<div style="border: 1px solid black; padding: 2px; display: inline-block;">1234</div>
Data Memory		
0300h	<div style="border: 1px solid black; padding: 2px; display: inline-block;">1234</div>	<div style="border: 1px solid black; padding: 2px; display: inline-block;">1234</div>

**Syntax** **MVDP** *Smem*, *pmad*

**Operands** *Smem*: Single data-memory operand  
 $0 \leq pmad \leq 65\,535$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** *pmad* → PAR  
 If (RC) ≠ 0  
 Then  
     (*Smem*) → Pmem addressed by PAR  
     (PAR) + 1 → PAR  
 Else  
     (*Smem*) → Pmem addressed by PAR

**Status Bits** None

**Description** This instruction copies a 16-bit single data-memory operand *Smem* to a program-memory location addressed by a 16-bit immediate value *pmad*. You can use this instruction with the repeat instruction to move consecutive words in data memory (using indirect addressing) to the contiguous program-memory space addressed by 16-bit immediate values. The source and destination blocks do not have to be entirely on-chip or off-chip. When used with repeat, this instruction becomes a single-cycle instruction after the repeat pipeline starts. In addition, when repeat is used with this instruction, interrupts are inhibited. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 4 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 20A (see page 3-44)  
 Class 20B (see page 3-46)

## **MVDP** *Move Data From Data Memory to Program Memory*

---

### **Example**

MVDP DAT0, 0FE00h

	Before Instruction	After Instruction
DP	<div>004</div>	<div>004</div>
Data Memory		
0200h	<div>0123</div>	<div>0123</div>
Program Memory		
FE00h	<div>FFFF</div>	<div>0123</div>



**Syntax** **MVKD** *dmad*, *Smem*

**Operands** *Smem*: Single data-memory operand  
 $0 \leq dmad \leq 65535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	0	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** *dmad* → DAR  
 If (RC) ≠ 0  
 Then  
     (Dmem addressed by DAR) → *Smem*  
     (DAR) + 1 → xDAR  
 Else  
     (Dmem addressed by DAR) → *Smem*

**Status Bits** None

**Description** This instruction moves data from data memory to data memory. The source data-memory value is addressed with a 16-bit immediate operand *dmad* and is moved to *Smem*. You can use this instruction with the single repeat instruction to move consecutive words in data memory (using indirect addressing). The number of words to move is one greater than the number contained in the repeat counter at the beginning of the instruction. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 19A (see page 3-40)  
 Class 19B (see page 3-42)

**Example 1** MVKD 300h, 0

	Before Instruction	After Instruction
DP	004	004
Data Memory		
0200h	ABCD	1234
0300h	1234	1234

## **MVKD** *Move Data From Data Memory to Data Memory With Source Addressing*

---

### **Example 2**

MVKD 1000h, \*+AR5

	Before Instruction	After Instruction
AR5	01FF	0200
Data Memory		
1000h	1234	1234
0200h	ABCD	1234

**Syntax** MVMD MMR, dmad

**Operands** MMR: Memory-mapped register  
 $0 \leq dmad \leq 65535$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	1	1	1	A	A	A	A	A	A	A
16-bit constant															

**Execution** dmad → EAR  
 If (RC) ≠ 0  
 Then  
     (MMR) → Dmem addressed by EAR  
     (EAR) + 1 → EAR  
 Else  
     (MMR) → Dmem addressed by EAR

**Status Bits** None

**Description** This instruction moves data from a memory-mapped register *MMR* to data memory. The data-memory destination is addressed with a 16-bit immediate value *dmad*. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

**Cycles** 2 cycles

**Classes** Class 19A (see page 3-40)

**Example** MVMD AR7, 8000h

	Before Instruction	After Instruction
AR7	1234	1234
Data Memory		
8000h	ABCD	1234

## MVMM *Move Data From Memory-Mapped Register to Memory-Mapped Register*

**Syntax** MVMM MMRx, MMRy

**Operands** MMRx: AR0–AR7, SP  
MMRy: AR0–AR7, SP

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	1	M	M	R	X	M	M	R	Y

Register	MMRX/MMRY	Register	MMRX/MMRY
AR0	0000	AR5	0101
AR1	0001	AR6	0110
AR2	0010	AR7	0111
AR3	0011	SP	1000
AR4	0100		

**Execution** (MMRx) → MMRy

**Status Bits** None

**Description** This instruction moves the content of memory-mapped register *MMRx* to the memory-mapped register *MMRy*. Only nine operands are allowed: AR0–AR7 and SP. The read operation from *MMRx* is executed in the decode phase. The write operation to *MMRy* is executed in the access phase.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example** MVMM SP, AR1

	Before Instruction		After Instruction
AR1	3EFF	AR1	0200
SP	0200	SP	0200

**Syntax** **MVPD** *pmad*, *Smem*

**Operands** *Smem*: Single-memory operand  
 $0 \leq pmad \leq 65\,535$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	I	A	A	A	A	A	A	A
16-bit constant															

**Execution** *pmad* → PAR  
 If (RC) ≠ 0  
 Then  
     (Pmem addressed by PAR) → *Smem*  
     (PAR) + 1 → PAR  
 Else  
     (Pmem addressed by PAR) → *Smem*

**Status Bits** None

**Description** This instruction moves a word in program memory addressed by a 16-bit immediate value *pmad* to a data-memory location addressed by *Smem*. This instruction can be used with the repeat instruction to move consecutive words addressed by a 16-bit immediate program address to contiguous data-memory locations addressed by *Smem*. The source and destination blocks do not have to be entirely on-chip or off-chip. When used with repeat, this instruction becomes a single-cycle instruction after the repeat pipeline starts. In addition, when repeat is used with this instruction, interrupts are inhibited. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 3 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 21A (see page 3-47)  
 Class 21B (see page 3-49)

## **MVPD** *Move Data From Program Memory to Data Memory*

### **Example 1**

MVPD 0FE00h, DAT5

Before Instruction		After Instruction	
DP	006	DP	006
Program Memory			
FE00h	8A55	FE00h	8A55
Data Memory			
0305h	FFFF	0305h	8A55

### **Example 2**

MVPD 2000h, \*AR7-0

Before Instruction		After Instruction	
AR0	0002	AR0	0002
AR7	0FFE	AR7	0FFC
Program Memory			
2000h	1234	2000h	1234
Data Memory			
0FFEh	ABCD	0FFEh	1234

**Syntax** **NEG** *src* [, *dst*]**Operands** *src, dst*: A (accumulator A)  
B (accumulator B)**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	0	0	1	0	0

**Execution** (*src*)  $\times -1 \rightarrow$  *dst***Status Bits** Affected by OVM  
Affects C and OV*dst* (or OV*src*, when *dst* = *src*)**Description** This instruction computes the 2s complement of the content of *src* (either A or B) and stores the result in *dst* or *src*, if *dst* is not specified. This instruction clears the carry bit, C, to 0 for all nonzero values of the accumulator. If the accumulator equals 0, the carry bit is set to 1.

If the accumulator equals 80 0000 0000h, the negate operation causes an overflow, because the 2s complement of 80 0000 0000h exceeds the capacity of the accumulator. As a result, if OVM = 1, then *dst* is assigned 00 7FFF FFFFh. If OVM = 0, *dst* is assigned 80 0000 0000h. The OV bit for *dst* is set to indicate overflow in either case.

**Words** 1 word**Cycles** 1 cycle**Classes** Class 1 (see page 3-3)**Example 1** **NEG** A, B

	Before Instruction	After Instruction
A	FF FFFF F228	FF FFFF F228
B	00 0000 1234	00 0000 0DD8
OVA	0	0

**Example 2** **NEG** B, A

	Before Instruction	After Instruction
A	00 0000 1234	FF 8000 0000
B	00 8000 0000	00 8000 0000
OVB	0	0

**Example 3** **NEG** A

	Before Instruction	After Instruction
A	80 0000 0000	80 0000 0000
OVA	0	1
OVM	0	0

**NEG** *Negate Accumulator*

---

**Example 4**            NEG A

Before Instruction		After Instruction	
A	80 0000 0000	A	00 7FFF FFFF
OVA	0	OVA	1
OVM	1	OVM	1



Syntax	NOP																																
Operands	None																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	0	1	0	0	1	0	1	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	1	1	1	0	1	0	0	1	0	0	1	0	1	0	1																		
Execution	None																																
Status Bits	None																																
Description	No operation is performed. Only the PC is incremented. This is useful to create pipeline and execution delays.																																
Words	1 word																																
Cycles	1 cycle																																
Classes	Class 1 (see page 3-3)																																
Example	NOP  No operation is performed.																																

## **NORM** *Normalization*

**Syntax** **NORM** *src* [, *dst*]

**Operands** *src, dst* : A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	0	1	1	1	1

**Execution** (*src*) << TS → *dst*

**Status Bits** Affected by SXM and OVM  
Affects OVdst (or OVsrc, when *dst* = *src*)

**Description** The signed number contained in *src* is normalized and the value is stored in *dst* or *src*, if *dst* is not specified. Normalizing a fixed-point number separates the number into a mantissa and an exponent by finding the magnitude of the sign-extended number.

This instruction allows single-cycle normalization of the accumulator once the EXP instruction, which computes the exponent of a number, has executed. The shift value is defined by T(5–0) and coded as a 2s-complement number. The valid shift values are –16 to 31. For the normalization, the shifter needs the shift value (in T) in the read phase; the normalization is executed in the execution phase.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1** NORM A

	Before Instruction	After Instruction
A	FF FFFF F001	FF 8008 0000
T	0013	0013

**Example 2** NORM B, A

	Before Instruction	After Instruction
A	FF FFFF F001	00 4214 1414
B	21 0A0A 0A0A	21 0A0A 0A0A
T	0FF9	0FF8

**Syntax**

- 1: **OR** *Smem*, *src*
- 2: **OR** *#lk* [, *SHFT*], *src* [, *dst*]
- 3: **OR** *#lk*, 16, *src* [, *dst*]
- 4: **OR** *src* [, *SHIFT*], [, *dst*]

**Operands**

*src*, *dst* : A (accumulator A)  
               B (accumulator B)  
*Smem* : Single data-memory operand  
 $0 \leq \text{SHFT} \leq 15$   
 $-16 \leq \text{SHIFT} \leq 15$   
 $0 \leq \text{lk} \leq 65535$

**Opcode**

1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	S	I	A	A	A	A	A	A	A

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	0	0	S	H	F	T
16-bit constant															

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	1	0	0
16-bit constant															

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	1	0	1	S	H	I	F	T

**Execution**

- 1: (*Smem*) OR (*src*(15–0)) → *src*  
*src*(39–16) unchanged
- 2: *lk* << *SHFT* OR (*src*) → *dst*
- 3: *lk* << 16 OR (*src*) → *dst*
- 4: (*src* or [*dst*]) OR (*src*) << *SHIFT* → *dst*

**Status Bits**

None

**Description**

This instruction ORs the *src* with a single data-memory operand *Smem*, a left-shifted 16-bit immediate value *lk*, *dst*, or with itself. The result is stored in *dst*, or *src* if *dst* is not specified. The values can be shifted as indicated by the instruction. For a positive (left) shift, low-order bits are cleared and high-order bits are not sign extended. For a negative (right) shift, high-order bits are not sign extended.

## **OR** *OR With Accumulator*

---

### **Words**

Syntaxes 1 and 4: 1 word  
Syntaxes 2 and 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

### **Cycles**

Syntaxes 1 and 4: 1 cycle  
Syntaxes 2 and 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

### **Classes**

Syntax 1: Class 3A (see page 3-5)  
Syntax 1: Class 3B (see page 3-6)  
Syntaxes 2 and 3: Class 2 (see page 3-4)  
Syntax 4: Class 1 (see page 3-3)

### **Example 1**

OR \*AR3+, A

		Before Instruction	After Instruction
A		00 00FF 1200	00 00FF 1700
AR3		0100	0101
Data Memory			
0100h		1500	1500

### **Example 2**

OR A, +3, B

		Before Instruction	After Instruction
A		00 0000 1200	00 0000 1200
B		00 0000 1800	00 0000 9800

**Syntax** **ORM** *#lk, Smem*

**Operands** Smem: Single data-memory operand  
 $0 \leq lk \leq 65\,535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	0	1	1	A	A	A	A	A	A	A
16-bit constant															

**Execution**  $lk \text{ OR } (Smem) \rightarrow Smem$

**Status Bits** None

**Description** This instruction ORs the single data-memory operand *Smem* with a 16-bit constant *lk*, and stores the result in *Smem*. This instruction is a memory-to-memory operation.

**Note:**

This instruction is not repeatable.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 18A (see page 3-39)  
 Class 18B (see page 3-39)

**Example** ORM 0404h, \*AR4+

	Before Instruction	After Instruction
AR4	0100	0101
Data Memory		
0100h	4444	4444

## POLY *Polynomial Evaluation*

**Syntax** POLY *Smem*

**Operands** *Smem* : Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	1	0	I	A	A	A	A	A	A	A

**Execution** Round  $(A(32-16) \times (T) + (B)) \rightarrow A$   
 $(Smem) \ll 16 \rightarrow B$

**Status Bits** Affected by FRCT, OVM, and SXM  
 Affects OVA

**Description** This instruction shifts the content of the single data-memory operand *Smem* 16 bits to the left and stores the result in accumulator B. In parallel, this instruction multiplies the high part of accumulator A (bits 32–16) by the content of T, adds the product to accumulator B, rounds the result of this operation, and stores the final result in accumulator A. This instruction is useful for polynomial evaluation to implement computations that take one cycle per monomial to execute.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 3A (see page 3-5)  
 Class 3B (see page 3-6)

**Example** POLY \*AR3+%

	Before Instruction	After Instruction
A	00 1234 0000	00 0627 0000
B	00 0001 0000	00 2000 0000
T	5678	5678
AR3	0200	0201
Data Memory		
0200h	2000	2000

**Syntax** **POPD** *Smem*

**Operands** *Smem*: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	1	A	A	A	A	A	A	A

**Execution**  
 (TOS) → *Smem*  
 (SP) + 1 → SP

**Status Bits** None

**Description** This instruction moves the content of the data-memory location addressed by SP to the memory location specified by *Smem*. SP is incremented by 1.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes**  
 Class 17A (see page 3-36)  
 Class 17B (see page 3-38)

**Example** POPD DAT10

	Before Instruction	After Instruction
DP	008	008
SP	0300	0301
Data Memory		
0300h	0092	0092
040Ah	0055	0092

## POPM *Pop Top of Stack to Memory-Mapped Register*

**Syntax** POPM *MMR*

**Operands** MMR: Memory-mapped register

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	1	A	A	A	A	A	A	A

**Execution** (TOS) → MMR  
(SP) + 1 → SP

**Status Bits** None

**Description** This instruction moves the content of the data-memory location addressed by SP to the specified memory-mapped register *MMR*. SP is incremented by 1.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 17A (see page 3-36)

**Example** POPM AR5

	Before Instruction	After Instruction
AR5	0055	0060
SP	03F0	03F1
Data Memory		
03F0h	0060	0060



**Syntax** **PORTR** *PA, Smem*

**Operands** Smem: Single data-memory operand  
 $0 \leq PA \leq 65\,535$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	0	I	A	A	A	A	A	A	A
Port address															

**Execution** (PA) → Smem

**Status Bits** None

**Description** This instruction reads a 16-bit value from an external I/O port *PA* (16-bit immediate address) into the specified data-memory location *Smem*. The  $\overline{IS}$  signal goes low to indicate an I/O access, and the  $\overline{IOSTRB}$  and  $READY$  timings are the same as for an external data memory read.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 2 cycles (dependent on the external I/O operation)

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 27A (see page 3-63)  
 Class 27B (see page 3-63)

**Example** PORTR 05, INDAT ; INDAT .equ 60h

Before Instruction		After Instruction	
DP	000	DP	000
I/O Memory			
0005h	7FFA	0005h	7FFA
Data Memory			
0060h	0000	0060h	7FFA

## PORTW Write Data to Port

**Syntax** PORTW *Smem*, *PA*

**Operands** Smem: Single data-memory operand  
 $0 \leq PA \leq 65\,535$

<b>Opcode</b>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	1	1	0	1	0	1	I	A	A	A	A	A	A	A
	Port address															

**Execution** (Smem) → PA

**Status Bits** None

**Description** This instruction writes a 16-bit value from the specified data-memory location *Smem* to an external I/O port *PA*. The  $\overline{IS}$  signal goes low to indicate an I/O access, and the  $\overline{IOSTRB}$  and READY timings are the same as for an external data memory read.

**Words** 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 2 cycles (dependent on the external I/O operation)

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 28A (see page 3-64)  
Class 28B (see page 3-65)

**Example** PORTW OUTDAT, 5h ; OUTDAT .equ 07h

	Before Instruction	After Instruction
DP	001	001
I/O Memory		
0005h	0000	7FFA
Data Memory		
0087h	7FFA	7FFA

**Syntax** PSHD *Smem***Operands** Smem: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	1	I	A	A	A	A	A	A	A

**Execution** (SP) – 1 → SP  
 (Smem) → TOS

**Status Bits** None

**Description** After SP has been decremented by 1, this instruction stores the content of the memory location *Smem* in the data-memory location addressed by SP. SP is read during the decode phase; it is stored during the access phase.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 16A (see page 3-33)  
 Class 16B (see page 3-35)

**Example** PSHD \*AR3+

	Before Instruction	After Instruction
AR3	0200	0201
SP	8000	7FFF
Data Memory		
0200h	07FF	07FF
7FFFh	0092	07FF

## PSHM *Push Memory-Mapped Register Onto Stack*

Syntax	PSHM <i>MMR</i>																																
Operands	MMR:    Memory-mapped register																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	0	0	1	0	1	0	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	0	0	1	0	1	0	I	A	A	A	A	A	A	A																		
Execution	(SP) – 1 → SP (MMR) → TOS																																
Status Bits	None																																
Description	After SP has been decremented by 1, this instruction stores the content of the memory-mapped register <i>MMR</i> in the data-memory location addressed by SP.																																
Words	1 word																																
Cycles	1 cycle																																
Classes	Class 16A (see page 3-33)																																
Example	PSHM BRC																																

	Before Instruction		After Instruction				
BRC	<table><tr><td></td><td>1234</td></tr></table>		1234	BRC	<table><tr><td></td><td>1234</td></tr></table>		1234
	1234						
	1234						
SP	<table><tr><td></td><td>2000</td></tr></table>		2000	SP	<table><tr><td></td><td>1FFF</td></tr></table>		1FFF
	2000						
	1FFF						
Data Memory							
1FFFh	<table><tr><td></td><td>07FF</td></tr></table>		07FF	1FFFh	<table><tr><td></td><td>1234</td></tr></table>		1234
	07FF						
	1234						

**Syntax** **RC[D]** *cond* [, *cond* [, *cond*]]

**Operands** The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{BIO}$ low	0000 0011	NBIO	$\overline{BIO}$ high	0000 0010
C	$C = 1$	0000 1100	NC	$C = 0$	0000 1000
TC	$TC = 1$	0011 0100	NTC	$TC = 0$	0010 0000
AEQ	$(A) = 0$	0100 0101	BEQ	$(B) = 0$	0100 1101
ANEQ	$(A) \neq 0$	0100 0100	BNEQ	$(B) \neq 0$	0100 1100
AGT	$(A) > 0$	0100 0110	BGT	$(B) > 0$	0100 1110
AGEQ	$(A) \geq 0$	0100 0010	BGEQ	$(B) \geq 0$	0100 1010
ALT	$(A) < 0$	0100 0011	BLT	$(B) < 0$	0100 1011
ALEQ	$(A) \leq 0$	0100 0111	BLEQ	$(B) \leq 0$	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	1	1	1	1	Z	0	C	C	C	C	C	C	C	C

**Execution** If (cond(s))  
 Then  
     (TOS)  $\rightarrow$  PC  
     (SP) + 1  $\rightarrow$  SP  
 Else  
     (PC) + 1  $\rightarrow$  PC

**Status Bits** None

**Description** If the conditions given by *cond* are met, this instruction replaces the PC with the data-memory value from the TOS and increments the SP by 1. If the conditions are not met, this instruction just increments the PC by 1.

If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed. The two instruction words following this instruction have no effect on the condition(s) being tested.

**RC[D]** *Return Conditionally*

This instruction tests multiple conditions before passing control to another section of the program. It can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

- Group 1 You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time.
- Group 2 You can select up to three conditions. Each of these conditions must be from a different category (category A, B, or C); you cannot have two conditions from the same category. For example, you can test TC, C, and BIO at the same time but you cannot test NTC, C, and NC at the same time.

Conditions for This Instruction				
Group 1		Group 2		
Category A	Category B	Category A	Category B	Category C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

**Note:**

This instruction is not repeatable.

<b>Words</b>	1 word
<b>Cycles</b>	5 cycles (true condition) 3 cycles (false condition) 3 cycles (delayed)
<b>Classes</b>	Class 32 (see page 3-70)

**Example**

RC AGEQ, ANOV ; return is executed if the accumulator A  
; contents are positive and the OVA bit  
; is a zero

	Before Instruction	After Instruction
PC	0807	2001
OVA	0	0
SP	0308	0309
Data Memory		
0308h	2002	2002

## **READA** *Read Program Memory Addressed by Accumulator A and Store in Data Memory*

**Syntax** **READA** *Smem*

**Operands** **Smem:** Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	0	1	A	A	A	A	A	A	A

**Execution**

$A \rightarrow \text{PAR}$   
If  $((\text{RC}) \neq 0)$   
     $(\text{Pmem}(\text{addressed by PAR})) \rightarrow \text{Smem}$   
     $(\text{PAR}) + 1 \rightarrow \text{PAR}$   
     $(\text{RC}) - 1 \rightarrow \text{RC}$   
Else  
     $(\text{Pmem}(\text{addressed by PAR})) \rightarrow \text{Smem}$

**Status Bits** None

**Description** This instruction transfers a word from a program-memory location defined by accumulator A to a data-memory location *Smem*. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction. Accumulator A defines the program-memory location according to the specific device, as follows:

'541–'546	'548
A(15–0)	A(22–0)

This instruction can be used with the repeat instruction to move consecutive words (starting with the address specified in accumulator A) to a contiguous data-memory space addressed using indirect addressing. Source and destination blocks do not need to be entirely on-chip or off-chip.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 5 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 25A (see page 3-57)  
Class 25B (see page 3-59)



*Read Program Memory Addressed by Accumulator A and Store in Data Memory*    **READA**

**Example**

READA DAT6

	Before Instruction	After Instruction
A	00 0000 0023	00 0000 0023
DP	004	004
Program Memory		
0023h	0306	0306
Data Memory		
0206h	0075	0306

## RESET *Software Reset*

**Syntax** RESET

**Operands** None

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	1	1	1	1	0	0	0	0	0

**Execution** These fields of PMST, ST0, and ST1 are loaded with the values shown:

(IPTR) << 7 → PC	0 → OVA	0 → OVB
1 → C	1 → TC	0 → ARP
0 → DP	1 → SXM	0 → ASM
0 → BRAF	0 → HM	1 → XF
0 → C16	0 → FRCT	0 → CMPT
0 → CPL	1 → INTM	0 → IFR
0 → OVM		

**Status Bits** The status bits affected are listed in the execution section.

**Description** This instruction performs a nonmaskable software reset that can be used at any time to put the '54x into a known state. When the reset instruction is executed, the operations listed in the execution section occur. The  $\overline{\text{MP/MC}}$  pin is not sampled during this software reset. The initialization of IPTR and the peripheral registers is different from the initialization using  $\overline{\text{RS}}$ . This instruction is not affected by INTM; however, it sets INTM to 1 to disable interrupts.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 3 cycles

**Classes** Class 35 (see page 3-72)

**Example** RESET

	Before Instruction		After Instruction
PC	0025	PC	0080
INTM	0	INTM	1
IPTR	1	IPTR	1

**Syntax** RET[D]**Operands** None

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	Z	0	0	0	0	0	0	0	0	0

**Execution**

(TOS) → PC  
(SP) + 1 → SP

**Status Bits** None

**Description**

This instruction replaces the value in the PC with the 16-bit value from the TOS. The SP is incremented by 1. If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles**

5 cycles  
3 cycles (delayed)

**Classes** Class 32 (see page 3-70)**Example** RET

	Before Instruction	After Instruction
PC	2112	1000
SP	0300	0301
Data Memory		
0300h	1000	1000

## RETE[D] *Enable Interrupts and Return From Interrupt*

**Syntax** RETE[D]

**Operands** None

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	0	1	1	1	0	1	0	1	1

**Execution** (TOS) → PC  
(SP) + 1 → SP  
0 → INTM

**Status Bits** Affects INTM

**Description** This instruction replaces the value in the PC with the 16-bit value from the TOS. Execution continues from this address. The SP is incremented by 1. This instruction automatically clears the interrupt mask bit (INTM) in ST1. (Clearing this bit enables interrupts.) If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 5 cycles  
3 cycles (delayed)

**Classes** Class 32 (see page 3-70)

**Example** RETE

	Before Instruction	After Instruction
PC	01C3	0110
SP	2001	2002
ST1	xCxx	x4xx
Data Memory		
2001h	0110	0110

**Syntax** RETF[D]

**Operands** None

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	Z	0	1	0	0	1	1	0	1	1

**Execution**  
 (RTN) → PC  
 (SP) + 1 → SP  
 0 → INTM

**Status Bits** Affects INTM

**Description** This instruction replaces the value in the PC with the 16-bit value in the RTN register. RTN holds the address to which the interrupt service routine should return. RTN is loaded into the PC during the return instead of reading the PC from the stack. The SP is incremented by 1. This instruction automatically clears the interrupt mask bit (INTM) in ST1. (Clearing this bit enables interrupts.) If the return is delayed (specified by the D suffix), the two 1-word instructions or one 2-word instruction following this instruction is fetched and executed.

**Note:**

You can use this instruction only if no call is performed during the interrupt service routine and no other interrupt routine is taken.

This instruction is not repeatable.

**Words** 1 word

**Cycles** 3 cycles  
 1 cycle (delayed)

**Classes** Class 33 (see page 3-71)

**Example** RETF

	Before Instruction		After Instruction
PC	01C3	PC	0110
SP	2001	SP	2002
ST1	xCxx	ST1	x4xx
Data Memory			
2001h	0110	2001h	0110

## RND Round Accumulator (LP Devices)

**Syntax** RND *src* [, *dst*]

**Operands** src , dst: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	1	1	1	1	1

**Execution** (src) + 8000h → dst

**Status Bits** Affected by OVM

**Description** This instruction rounds the content of *src* (either A or B) by adding  $2^{15}$ . The rounded value is stored in *dst* or *src*, if *dst* is not specified.

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example** RND A, B

	Before Instruction	After Instruction
A	FF FFFF FFFF	0F FFFF FFFF
B	00 0000 0001	00 0000 7FFF
OVM	0	0

**Example** RND A

	Before Instruction	After Instruction
A	00 7FFF FFFF	00 7FFF 7FFF
OVM	1	1

**Syntax**                    **ROL** *src*

**Operands**                *src* :    A (accumulator A)  
                                       B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	S	1	0	0	1	0	0	0	1

**Execution**                (C) → *src*(0)  
                                      (*src*(30–0)) → *src*(31–1)  
                                      (*src*(31)) → C  
                                      0 → *src*(39–32)

**Status Bits**                Affected by C  
                                      Affects C

**Description**                This instruction rotates each bit of *src* left 1 bit. The value of the carry bit, C, before the execution of the instruction is shifted into the LSB of *src*. Then, the MSB of *src* is shifted into C. The guard bits of *src* are cleared.

**Words**                      1 word

**Cycles**                     1 cycle

**Classes**                    Class 1 (see page 3-3)

**Example**                    ROL A

	Before Instruction		After Instruction
A	5F B000 1234	A	00 6000 2468
C	0	C	1

## ROLTC *Rotate Accumulator Left Using TC*

### Syntax

**ROLTC** *src*

### Operands

src: A (accumulator A)  
B (accumulator B)

### Opcode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	S	1	0	0	1	0	0	1	0

### Execution

(TC) → src(0)  
(src(30–0)) → src(31–1)  
(src(31)) → C  
0 → src(39–32)

### Status Bits

Affects C  
Affected by TC

### Description

This instruction rotates each bit of *src* left 1 bit. The value of the TC bit before the execution of the instruction is shifted into the LSB of *src*. Then, the MSB of *src* is shifted into C. The guard bits of *src* are cleared.

### Words

1 word

### Cycles

1 cycle

### Classes

Class 1 (see page 3-3)

### Example

ROLTC A

	Before Instruction		After Instruction
A	81 C000 5555	A	00 8000 AAAB
C	x	C	1
TC	1	TC	1



**Syntax** **ROR** *src*

**Operands** *src*: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	S	1	0	0	1	0	0	0	0

**Execution**  
 (C) → *src*(31)  
*src*(31-1) → *src*(30-0)  
*src*(0) → C  
 0 → *src*(39-32)

**Status Bits**  
 Affects C  
 Affected by C

**Description** This instruction rotates each bit of *src* right 1 bit. The value of the carry bit, C, before the execution of the instruction is shifted into the MSB of *src*. Then, the LSB of *src* is shifted into C. The guard bits of *src* are cleared.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example** ROR A

	Before Instruction	After Instruction
A	7F 8000 1235	00 5800 091A
C	0	1

## RPT Repeat Next Instruction

**Syntax**

- 1: **RPT** *Smem*
- 2: **RPT** *#K*
- 3: **RPT** *#lk*

**Operands**

*Smem*: Single data-memory operand  
 $0 \leq K \leq 255$   
 $0 \leq lk \leq 65\,535$

**Opcode**

1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1	A	A	A	A	A	A	A

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0	K	K	K	K	K	K	K	K

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
16-bit constant															

**Execution**

- 1: (*Smem*)  $\rightarrow$  RC
- 2: *K*  $\rightarrow$  RC
- 3: *lk*  $\rightarrow$  RC

**Status Bits** None

**Description**

The repeat counter (RC) is loaded with the number of iterations when this instruction is executed. The number of iterations (*n*) is given in a 16-bit single data-memory operand *Smem* or an 8- or 16-bit constant, *K* or *lk*, respectively. The instruction following the repeat instruction is repeated *n* + 1 times. You cannot access RC while it decrements.

**Note:**

This instruction is not repeatable.

**Words**

Syntaxes 1 and 2: 1 word  
Syntax 3: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles**

Syntaxes 1 and 2: 1 cycle  
Syntax 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes**

Syntax 1: Class 5A (see page 3-9)

Syntax 1: Class 5B (see page 3-9)

Syntax 2: Class 1 (see page 3-3)

Syntax 3: Class 2 (see page 3-4)

**Example 1**

RPT DAT127 ; DAT127 .EQU 0FFFh

Before Instruction		After Instruction	
RC	<input type="text" value="0"/>	RC	<input type="text" value="000C"/>
DP	<input type="text" value="031"/>	DP	<input type="text" value="031"/>
Data Memory			
0FFFh	<input type="text" value="000C"/>	0FFFh	<input type="text" value="000C"/>

**Example 2**

RPT #2 ; Repeat next instruction 3 times

Before Instruction		After Instruction	
RC	<input type="text" value="0"/>	RC	<input type="text" value="0002"/>

**Example 3**

RPT #1111h ; Repeat next instruction 4370 times

Before Instruction		After Instruction	
RC	<input type="text" value="0"/>	RC	<input type="text" value="1111"/>

## RPTB[D] Block Repeat

**Syntax** RPTB[D] *pmad*

**Operands**  $0 \leq pmad \leq 65\,535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	Z	0	0	1	1	1	0	0	1	0
16-bit constant															

**Execution**

1 → BRAF  
If (delayed) then  
    (PC) + 4 → RSA  
Else  
    (PC) + 2 → RSA  
*pmad* → REA

**Status Bits** Affects BRAF

**Description** This instruction repeats a block of instructions the number of times specified by the memory-mapped block-repeat counter (BRC). BRC must be loaded before the execution of this instruction. When this instruction is executed, the block-repeat start address register (RSA) is loaded with PC + 2 (or PC + 4 if you use the delayed instruction) and the block-repeat end address register (REA) is loaded with the program-memory address (*pmad*).

This instruction is interruptible. Single-instruction repeat loops can be included as part of block repeat blocks. To nest instructions you must ensure that:

- ☐ The BRC, RSA, and REA registers are appropriately saved and restored.
- ☐ The block-repeat active flag (BRAF) is properly set.

In a delayed block repeat (specified by the D suffix), the two 1-word instructions or the one 2-word instruction following this instruction is fetched and executed.

**Note:**

Block repeat can be deactivated by clearing the BRAF bit.

This instruction is not repeatable.

**Words** 2 words

**Cycles** 4 cycles  
2 cycles (delayed)

**Classes** Class 29A (see page 3-66)

**Example 1**

```
ST #99, BRC
RPTB end_block - 1
; end_block = Bottom of Block
```

Before Instruction		After Instruction	
PC	1000	PC	1001
BRC	1234	BRC	0063
RSA	5678	RSA	1002
REA	9ABC	REA	9ABC

**Example 2**

```
ST #99, BRC ;execute the block 100 times
RPTBD end_block - 1
MVDM POINTER, AR1
```

```
; initialize pointer
; end_block ; Bottom of Block
```

Before Instruction		After Instruction	
BRC	1234	BRC	0063
RSA	5678	RSA	1004
REA	9ABC	REA	9ABC

## RPTZ Repeat Next Instruction And Clear Accumulator

**Syntax** RPTZ *dst, #lk*

**Operands** dst: A (accumulator A)  
B (accumulator B)  
 $0 \leq lk \leq 65\,535$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	D	0	1	1	1	0	0	0	1
16-bit constant															

**Execution** 0 → dst  
lk → RC

**Status Bits** None

**Description** This instruction clears *dst* and repeats the next instruction  $n + 1$  times, where  $n$  is the value in the repeat counter (RC). The RC value is obtained from the 16-bit constant *lk*.

**Words** 2 words

**Cycles** 2 cycles

**Classes** Class 2 (see page 3-4)

**Example** RPTZ A, 1023 ; Repeat the next instruction 1024 times  
STL A, \*AR2+

Before Instruction		After Instruction	
A	0F FE00 8000	A	00 0000 0000
RC	0000	RC	03FF

**Syntax** **RSBX *N*, *SBIT*****Operands**  $0 \leq \text{SBIT} \leq 15$   
 $N = 0 \text{ or } 1$ **Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	N	0	1	0	1	1	S	B	I	T

**Execution**  $0 \rightarrow \text{STN}(\text{SBIT})$ **Status Bits** None**Description** This instruction clears the specified bit in status register 0 or 1 to a logic 0. *N* designates the status register to modify and *SBIT* specifies the bit to be modified. The name of a field in a status register can be used as an operand instead of the *N* and *SBIT* operands (see Example1).**Note:**

This instruction is not repeatable.

**Words** 1 word**Cycles** 1 cycle**Classes** Class 1 (see page 3-3)**Example 1** RSBX SXM ; SXM means: n=1 and SBIT=8

	Before Instruction	After Instruction
ST1	35CD	34CD

**Example 2** RSBX 1,8

	Before Instruction	After Instruction
ST1	35CD	34CD

**SACCD** *Store Accumulator Conditionally***Syntax** **SACCD** *src, Xmem, cond***Operands**  
**src:** A (accumulator A)  
B (accumulator B)  
**Xmem:** Dual data-memory operand

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) $\neq$ 0	0100	BNEQ	(B) $\neq$ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) $\geq$ 0	0010	BGEQ	(B) $\geq$ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) $\leq$ 0	0111	BLEQ	(B) $\leq$ 0	1111

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	1	S	X	X	X	X	C	O	N	D

**Execution**  
If (*cond*)  
Then  
    (*src*) << (ASM – 16) → Xmem  
Else  
    (Xmem) → (Xmem)**Status Bits** Affected by ASM and SXM**Description**  
If the condition is true, this instruction stores *src* left-shifted by (ASM – 16). The shift value is in the memory location designated by *Xmem*. If the condition is false, the instruction reads *Xmem* and writes the value in *Xmem* back to the same address; thus, *Xmem* remains the same. Regardless of the condition, *Xmem* is always read and updated.**Words** 1 word**Cycles** 1 cycle**Classes** Class 15 (see page 3-32)



**Example**

SACCD A, \*AR3+0%, ALT

Before Instruction		After Instruction	
A	FF FE00 4321	A	FF FE00 4321
ASM	01	ASM	01
AR0	0002	AR0	0002
AR3	0202	AR3	0204
Data Memory			
0202h	0101	0202h	FC00

## SAT Saturate Accumulator

**Syntax** SAT src

**Operands** src: A (accumulator A)  
B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	S	1	0	0	0	0	0	1	1

**Execution** Saturate (src) → src

**Status Bits** Affects OVsrc

**Description** Regardless of the OVM value, this instruction allows the saturation of the content of src on 32 bits.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1** SAT B

	Before Instruction	After Instruction
B	71 2345 6789	00 7FFF FFFF
OVb	x	1

**Example 2** SAT A

	Before Instruction	After Instruction
A	F8 1234 5678	FF 8000 0000
OvA	x	1

**Example 3** SAT B

	Before Instruction	After Instruction
B	00 0012 3456	00 0012 3456
OVb	x	0

Syntax	SFTA <i>src</i> , <i>SHIFT</i> [, <i>dst</i> ]																																
Operands	<i>src</i> , <i>dst</i> A (accumulator A) B (accumulator B) -16 ≤ <i>SHIFT</i> ≤ 15																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>S</td><td>D</td><td>0</td><td>1</td><td>1</td><td>S</td><td>H</td><td>I</td><td>F</td><td>T</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	S	D	0	1	1	S	H	I	F	T
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	1	1	1	0	1	S	D	0	1	1	S	H	I	F	T																		
Execution	<p>If <i>SHIFT</i> &lt; 0 Then     (<i>src</i>((-<i>SHIFT</i>) - 1)) → C     (<i>src</i>(39-0)) &lt;&lt; <i>SHIFT</i> → <i>dst</i>     If <i>SXM</i> = 1     Then         (<i>src</i>(39)) → <i>dst</i>(39-(39 + (<i>SHIFT</i> + 1))) (or <i>src</i>(39-(39 + (<i>SHIFT</i> + 1))),         if <i>dst</i> is not specified)     Else         0 → <i>dst</i>(39-(39 + (<i>SHIFT</i> + 1))) (or <i>src</i>(39-(39 + (<i>SHIFT</i> + 1))),         if <i>dst</i> is not specified) Else     (<i>src</i>(39 - <i>SHIFT</i>)) → C     (<i>src</i>) &lt;&lt; <i>SHIFT</i> → <i>dst</i>     0 → <i>dst</i>((<i>SHIFT</i> - 1)-0) (or <i>src</i>((<i>SHIFT</i> - 1)-0), if <i>dst</i> is not specified)</p>																																
Status Bits	<p>Affected by <i>SXM</i> and <i>OVM</i> Affects C and <i>OVdst</i> (or <i>OVsrc</i>, if <i>dst</i> = <i>src</i>)</p>																																
Description	<p>This instruction arithmetically shifts <i>src</i> and stores the result in <i>dst</i> or <i>src</i>, if <i>dst</i> is not specified. The execution of the instruction depends on the <i>SHIFT</i> value:</p> <div><div><input type="checkbox"/> If the <i>SHIFT</i> value is less than 0, the following occurs:</div><div><div>1) <i>src</i>((-<i>SHIFT</i>) - 1) is copied into the carry bit, C.</div><div>2) If <i>SXM</i> is 1, the instruction executes an arithmetic right shift and the MSB of the <i>src</i> is shifted into <i>dst</i>(39-(39 + (<i>SHIFT</i> + 1))).</div><div>3) If <i>SXM</i> is 0, 0 is written into <i>dst</i>(39-(39 + (<i>SHIFT</i> + 1))).</div></div><div><div><input type="checkbox"/> If the <i>SHIFT</i> value is greater than 0, the following occurs:</div><div><div>1) <i>src</i>(39 - <i>SHIFT</i>) is copied into the carry bit, C.</div><div>2) An arithmetic left shift is produced by the instruction.</div><div>3) 0 is written into <i>dst</i>((<i>SHIFT</i> - 1)-0).</div></div></div></div>																																
Words	1 word																																
Cycles	1 cycle																																

## SFTA *Shift Accumulator Arithmetically*

### Classes

Class 1 (see page 3-3)

### Example 1

SFTA A, -5, B

Before Instruction		After Instruction	
A	FF 8765 0055	A	FF 8765 0055
B	00 4321 1234	B	FF FC3B 2802
C	x	C	1
SXM	1	SXM	1

### Example 2

SFTA B, +5

Before Instruction		After Instruction	
B	80 AA00 1234	B	15 4002 4680
C	0	C	1
OVM	0	OVM	0
SXM	0	SXM	0

**Syntax** **SFTC** *src***Operands** **src:** A (accumulator A)  
B (accumulator B)**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	S	1	0	0	1	0	1	0	0

**Execution**

If (src) = 0  
 Then  
   1 → TC  
 Else  
   If (src(31)) XOR (src(30)) = 0  
   Then (two significant sign bits)  
     0 → TC  
     (src) << 1 → src  
   Else (only one sign bit)  
     1 → TC

**Status Bits** Affects TC

**Description** If *src* has two significant sign bits, this instruction shifts the 32-bit *src* left by 1 bit. If there are two sign bits, the test control (TC) bit is cleared to 0; otherwise, it is set to 1.

**Words** 1 word**Cycles** 1 cycle**Classes** Class 1 (see page 3-3)**Example** SFTC A

Before Instruction		After Instruction	
A	FF FFFF F001	A	FF FFFF E002
TC	x	TC	0

## SFTL *Shift Accumulator Logically*

**Syntax** SFTL *src*, *SHIFT* [, *dst*]

**Operands** src, dst: A (accumulator A)  
B (accumulator B)  
 $-16 \leq \text{SHIFT} \leq 15$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	1	1	1	S	H	I	F	T

**Execution**

If  $\text{SHIFT} < 0$   
Then  
src( $(-\text{SHIFT}) - 1$ )  $\rightarrow$  C  
src( $31-0$ )  $\ll$   $\text{SHIFT} \rightarrow$  dst  
0  $\rightarrow$  dst( $39-(31 + (\text{SHIFT} + 1))$ )  
If  $\text{SHIFT} = 0$   
Then  
0  $\rightarrow$  C  
Else  
src( $31 - (\text{SHIFT} - 1)$ )  $\rightarrow$  C  
src( $(31 - \text{SHIFT})-0$ )  $\ll$   $\text{SHIFT} \rightarrow$  dst  
0  $\rightarrow$  dst( $(\text{SHIFT} - 1)-0$ ) (or src( $(\text{SHIFT} - 1)-0$ ), if dst is not specified)  
0  $\rightarrow$  dst( $39-32$ ) (or src( $39-32$ ), if dst is not specified)

**Status Bits** Affects C

**Description** This instruction logically shifts *src* and stores the result in *dst* or *src*, if *dst* is not specified. The guard bits of *dst* or *src*, if *dst* is not specified, are also cleared. The execution of the instruction depends on the SHIFT value:

- ☐ If the SHIFT value is less than 0, the following occurs:
  - 1) src( $(-\text{SHIFT}) - 1$ ) is copied into the carry bit, C.
  - 2) A logical right shift is produced by the instruction.
  - 3) 0 is written into dst( $39-(31 + (\text{SHIFT} + 1))$ ).
- ☐ If the SHIFT value is greater than 0, the following occurs:
  - 1) src( $31 - (\text{SHIFT} - 1)$ ) is copied into the carry bit, C.
  - 2) A logical left shift is produced by the instruction.
  - 3) 0 is written into dst( $(\text{SHIFT} - 1)-0$ ).

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1**

SFTL A, -5, B

Before Instruction		After Instruction	
A	FF 8765 0055	A	FF 8765 0055
B	FF 8000 0000	B	00 043B 2802
C	0	C	1

**Example 2**

SFTL B, +5

Before Instruction		After Instruction	
B	80 AA00 1234	B	00 4002 4680
C	0	C	1

## SQDST *Square Distance*

**Syntax**                    **SQDST** *Xmem, Ymem*

**Operands**                *Xmem, Ymem*:    Dual data-memory operands

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	0	X	X	X	X	Y	Y	Y	Y

**Execution**                 $(A(32-16)) \times (A(32-16)) + (B) \rightarrow B$   
 $((Xmem) - (Ymem)) \ll 16 \rightarrow A$

**Status Bits**              Affected by OVM, FRCT, and SXM  
Affects C, INTM, OVA, and OVB

**Description**              Used in repeat single mode, this instruction computes the square of the distance between two vectors. The high part of accumulator A (bits 32–16) is squared, the product is added to accumulator B, and the result is stored in accumulator B. *Ymem* is subtracted from *Xmem*, the difference is shifted 16 bits left, and the result is stored in accumulator A. The value to be squared (*A(32–16)*) is the value of the accumulator before the subtraction is executed by this instruction.

**Words**                    1 word

**Cycles**                   1 cycle

**Classes**                  Class 7 (see page 3-12)

**Example**                  SQDST \*AR3+, AR4+

Before Instruction		After Instruction	
A	FF ABCD 0000	A	FF FFAB 0000
B	00 0000 0000	B	00 1BB1 8229
FRCT	0	FRCT	0
AR3	0100	AR3	0101
AR4	0200	AR4	0201
Data Memory			
0100h	0055	0100h	0055
0200h	00AA	0200h	00AA



Syntax	1: <b>SQUR</b> <i>Smem</i> , <i>dst</i> 2: <b>SQUR</b> <b>A</b> , <i>dst</i>																																																																
Operands	Smem: Single data-memory operand dst: A (accumulator A) B (accumulator B)																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>D</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>D</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	0	0	1	1	D	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	1	0	D	1	0	0	0	1	1	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
0	0	1	0	0	1	1	D	I	A	A	A	A	A	A	A																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
1	1	1	1	0	1	0	D	1	0	0	0	1	1	0	1																																																		
Execution	1: (Smem) → T (Smem) × (Smem) → dst 2: (A(32–16)) × (A(32–16)) → dst																																																																
Status Bits	Affected by OVM and FRCT Affects OVsrc																																																																
Description	This instruction squares a single data-memory operand <i>Smem</i> or the high part of accumulator A (bits 32–16) and stores the result in <i>dst</i> . T is unaffected when accumulator A is used; otherwise, <i>Smem</i> is stored in T.																																																																
Words	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																																																
Cycles	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																																																
Classes	Syntax 1: Class 3A (see page 3-5) Syntax 1: Class 3B (see page 3-6) Syntax 2: Class 1 (see page 3-3)																																																																

**Example 1**

SQUR DAT30, B

	Before Instruction	After Instruction
B	00 0000 01F4	00 0000 00E1
T	0003	000F
FRCT	0	0
DP	006	006
Data Memory		
031Eh	000F	000F

## SQUR *Square*

---

### Example 2

SQUR A, B

Before Instruction	
A	00 000F 0000
B	00 0101 0101
FRCT	1

After Instruction	
A	00 000F 0000
B	00 0000 01C2
FRCT	1

<b>Syntax</b>	<b>SQURA</b> <i>Smem</i> , <i>src</i>																																
<b>Operands</b>	Smem: Single data-memory operand src: A (accumulator A) B (accumulator B)																																
<b>Opcode</b>	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	1	1	1	0	0	S	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	0	1	1	1	0	0	S	I	A	A	A	A	A	A	A																		
<b>Execution</b>	(Smem) → T (Smem) × (Smem) + (src) → src																																
<b>Status Bits</b>	Affected by OVM and FRCT Affects OVsrc																																
<b>Description</b>	This instruction stores the data-memory value <i>Smem</i> in T, then it squares <i>Smem</i> and adds the product to <i>src</i> . The result is stored in <i>src</i> .																																
<b>Words</b>	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.																																
<b>Cycles</b>	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.																																
<b>Classes</b>	Class 3A (see page 3-5) Class 3B (see page 3-6)																																
<b>Example 1</b>	SQURA DAT30, B																																

	Before Instruction	After Instruction
B	00 0320 0000	00 0320 00E1
T	0003	000F
FRCT	0	0
DP	006	006
Data Memory		
031Eh	000F	000F

**Example 2**

SQURA \*AR3+, A

	Before Instruction	After Instruction
A	00 0000 01F4	00 0000 02D5
T	0003	000F
FRCT	0	0
AR3	031E	031F
Data Memory		
031Eh	000F	000F

## SQURS *Square and Subtract*

**Syntax** **SQURS** *Smem, src*

**Operands** **Smem:** Single data-memory operand  
**src:** A (accumulator A)  
 B (accumulator B)

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	S	I	A	A	A	A	A	A	A

**Execution** (Smem) → T  
 (src) – (Smem) × (Smem) → src

**Status Bits** Affected by OVM and FRCT  
 Affects OVsrc

**Description** This instruction stores the data-memory value *Smem* in T, then it squares *Smem* and subtracts the product from *src*. The result is stored in *src*.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 3A (see page 3-5)  
 Class 3B (see page 3-6)

**Example 1** SQURS DAT9, A

	Before Instruction	After Instruction
A	00 014B 5DB0	00 0000 0320
T	8765	1234
FRCT	0	0
DP	006	006
Data Memory		
0309h	1234	1234

**Example 2** SQURS \*AR3, B

	Before Instruction	After Instruction
B	00 014B 5DB0	00 0000 0320
T	8765	1234
FRCT	0	0
AR3	0309	0309
Data Memory		
0309h	1234	1234

**Syntax** **SRCCD** *Xmem, cond***Operands** *Xmem*: Dual data-memory operandThe following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) $\neq$ 0	0100	BNEQ	(B) $\neq$ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) $\geq$ 0	0010	BGEQ	(B) $\geq$ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) $\leq$ 0	0111	BLEQ	(B) $\leq$ 0	1111

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	1	X	X	X	X	C	O	N	D

**Execution**If (*cond*)

Then

(BRC)  $\rightarrow$  *Xmem*

Else

(Xmem)  $\rightarrow$  *Xmem***Status Bits**

None

**Description**

If the condition is true, this instruction stores the content of the block-repeat counter (BRC) in *Xmem*. If the condition is false, the instruction reads *Xmem* and writes the value in *Xmem* back to the same address; thus, *Xmem* remains the same. Regardless of the condition, *Xmem* is always read and updated.

**Words**

1 word

**Cycles**

1 cycle

**Classes**

Class 15 (see page 3-32)

**Example**

SRCCD \*AR5-, AGT

Before Instruction		After Instruction	
A	00 70FF FFFF	A	00 70FF FFFF
AR5	0202	AR5	0201
BRC	4321	BRC	4321
Data Memory			
0202h	1234	0202h	4321

## SSBX *Set Status Register Bit*

**Syntax** SSBX *N*, *SBIT*

**Operands**  $0 \leq \text{SBIT} \leq 15$   
 $N = 0 \text{ or } 1$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	N	1	1	0	1	1	S	B	I	T

**Execution**  $1 \rightarrow \text{STN}(\text{SBIT})$

**Status Bits** None

**Description** This instruction sets the specified bit in status register 0 or 1 to a logic 1. *N* designates the status register to modify and *SBIT* specifies the bit to be modified. The name of a field in a status register can be used as an operand instead of the *N* and *SBIT* operands (see Example 1).

**Note:**

This instruction is not repeatable.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 1 (see page 3-3)

**Example 1** SSBX *SXM* ; *SXM* means:  $N=1$ ,  $\text{SBIT}=8$

	Before Instruction	After Instruction
ST1	34CD	35CD

**Example 2** SSBX 1, 8

	Before Instruction	After Instruction
ST1	34CD	35CD

Syntax	1: ST T, Smem 2: ST TRN, Smem 3: ST #lk, Smem																																																																																																																
Operands	Smem: Single data-memory operand -32 768 ≤ lk ≤ 32 767																																																																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 3: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr><tr><td colspan="16">16-bit constant</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	0	0	1	1	0	0	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	0	0	0	1	1	0	1	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	1	0	1	1	0	I	A	A	A	A	A	A	A	16-bit constant															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																		
1	0	0	0	1	1	0	0	I	A	A	A	A	A	A	A																																																																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																		
1	0	0	0	1	1	0	1	I	A	A	A	A	A	A	A																																																																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																		
0	1	1	1	0	1	1	0	I	A	A	A	A	A	A	A																																																																																																		
16-bit constant																																																																																																																	
Execution	1: (T) → Smem 2: (TRN) → Smem 3: lk → Smem																																																																																																																
Status Bits	None																																																																																																																
Description	This instruction stores the content of T, the transition (TRN) register, or a 16-bit constant lk in data-memory location Smem.																																																																																																																
Words	Syntaxes 1 and 2: 1 word Syntax 3: 2 words  Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.																																																																																																																
Cycles	Syntaxes 1 and 2: 1 cycle Syntax 3: 2 cycles  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.																																																																																																																
Classes	Syntaxes 1 and 2: Class 10A (see page 3-22) Syntaxes 1 and 2: Class 10B (see page 3-23) Syntax 3: Class 12A (see page 3-26) Syntax 3: Class 12B (see page 3-27)																																																																																																																

## **ST** Store T, TRN, or Immediate Value Into Memory

### **Example 1**

ST FFFFh, DAT0

Before Instruction		After Instruction	
DP	004	DP	004
Data Memory			
0200h	0101	0200h	FFFF

### **Example 2**

ST TRN, DAT5

Before Instruction		After Instruction	
DP	004	DP	004
TRN	1234	TRN	1234
Data Memory			
0205h	0030	0205h	1234

### **Example 3**

ST T, \*AR7-

Before Instruction		After Instruction	
T	4210	T	4210
AR7	0321	AR7	0320
Data Memory			
0321h	1200	0321h	4210



**Syntax**

- 1: **STH** *src*, *Smem*
- 2: **STH** *src*, **ASM**, *Smem*
- 3: **STH** *src*, **SHFT**, *Xmem*
- 4: **STH** *src* [, **SHIFT**], *Smem*

**Operands**

src:           A (accumulator A)  
               B (accumulator B)  
 Smem:       Single data-memory operand  
 Xmem:       Dual data-memory operand  
 $0 \leq \text{SHFT} \leq 15$   
 $-16 \leq \text{SHIFT} \leq 15$

**Opcode**

1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	S	I	A	A	A	A	A	A	A

2:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	1	1	S	I	A	A	A	A	A	A	A

3:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	0	1	S	X	X	X	X	S	H	F	T

4:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	1	1	1	I	A	A	A	A	A	A	A
0	0	0	0	1	1	0	S	0	1	1	S	H	I	F	T

**Execution**

- 1: (*src*(31–16)) → *Smem*
- 2: (*src*) << (ASM – 16) → *Smem*
- 3: (*src*) << (SHFT – 16) → *Xmem*
- 4: (*src*) << (SHIFT – 16) → *Smem*

**Status Bits**

Affected by SXM

**Description**

This instruction stores the high part of *src* (bits 31–16) in data-memory location *Smem*. The *src* is shifted left (as specified by ASM, SHFT, or SHIFT) and bits 31–16 of the shifted value are stored in data memory (*Smem* or *Xmem*). If SXM = 0, bit 39 of *src* is copied in the MSBs of the data-memory location. If SXM = 1, the sign-extended value with bit 39 of *src* is stored in the MSBs of the data-memory location after being right-shifted by the exceeding guard bit margin. The *src* remains unaffected.

## STH Store Accumulator High Into Memory

### Notes:

The following syntaxes are assembled as a different syntax in certain cases.

- ☐ Syntax 3: If  $SHIFT = 0$ , the instruction opcode is assembled as syntax 1.
- ☐ Syntax 4: If  $SHIFT = 0$ , the instruction opcode is assembled as syntax 1.
- ☐ Syntax 4: If  $0 < SHIFT \leq 15$  and an indirect modifier is equal to one of the Xmem modes, the instruction opcode is assembled as syntax 3.

### Words

Syntaxes 1, 2, and 3: 1 word  
Syntax 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

### Cycles

Syntaxes 1, 2, and 3: 1 cycle  
Syntax 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

### Classes

Syntaxes 1, 2, and 3: Class 10A (see page 3-22)  
Syntaxes 1 and 2: Class 10B (see page 3-23)  
Syntax 4: Class 11A (see page 3-24)  
Syntax 4: Class 11B (see page 3-25)

### Example 1

STH A, DAT10

Before Instruction		After Instruction	
A	FF 8765 4321	A	FF 8765 4321
DP	004	DP	004
Data Memory			
020Ah	1234	020Ah	8765

### Example 2

STH B, -8, \*AR7-

Before Instruction		After Instruction	
B	FF 8421 1234	B	FF 8421 1234
AR7	0321	AR7	0320
Data Memory			
0321h	ABCD	0321h	FF84

**Example 3**

STH A, -4, DAT10

	Before Instruction	After Instruction
A	FF 8421 1234	FF 8421 1234
SXM	1	1
DP	004	004
Data Memory		
020Ah	7FFF	F842

## STL Store Accumulator Low Into Memory

### Syntax

- 1: **STL** *src*, *Smem*
- 2: **STL** *src*, **ASM**, *Smem*
- 3: **STL** *src*, **SHFT**, *Xmem*
- 4: **STL** *src* [, **SHIFT**], *Smem*

### Operands

*src*:           A (accumulator A)  
                B (accumulator B)  
*Smem*:         Single data-memory operand  
*Xmem*:         Dual data-memory operand  
 $0 \leq \text{SHFT} \leq 15$   
 $-16 \leq \text{SHIFT} \leq 15$

### Opcode

- 1:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 0  | 0  | 0  | 0  | 0  | 0 | S | I | A | A | A | A | A | A | A |
- 2:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 0  | 0  | 0  | 0  | 1  | 0 | S | I | A | A | A | A | A | A | A |
- 3:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 0  | 0  | 1  | 1  | 0  | 0 | S | X | X | X | X | S | H | F | T |
- 4:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 1  | 1  | 0  | 1  | 1  | 1 | 1 | I | A | A | A | A | A | A | A |
| 0  | 0  | 0  | 0  | 1  | 1  | 0 | S | 1 | 0 | 0 | S | H | I | F | T |

### Execution

- 1: (*src*(15–0)) → *Smem*
- 2: (*src*) << ASM → *Smem*
- 3: (*src*) << SHFT → *Xmem*
- 4: (*src*) << SHIFT → *Smem*

### Status Bits

Affected by SXM

### Description

This instruction stores the low part of *src* (bits 15–0) in data-memory location *Smem*. The *src* is shifted left (as specified by ASM, SHFT, or SHIFT) and bits 15–0 of the shifted value are stored in data memory (*Smem* or *Xmem*). When the shifted value is positive, zeros are shifted into the LSBs.

**Notes:**

The following syntaxes are assembled as a different syntax in certain cases.

- ☐ Syntax 3: If  $SHIFT = 0$ , the instruction opcode is assembled as syntax 1.
- ☐ Syntax 4: If  $SHIFT = 0$ , the instruction opcode is assembled as syntax 1.
- ☐ Syntax 4: If  $0 < SHIFT \leq 15$  and an indirect modifier is equal to one of the Xmem modes, the instruction opcode is assembled as syntax 3.

**Words**

Syntaxes 1, 2, and 3: 1 word

Syntax 4: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles**

Syntaxes 1, 2, and 3: 1 cycle

Syntax 4: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes**

Syntaxes 1, 2, and 3: Class 10A (see page 3-22)

Syntaxes 1, 2, and 3: Class 10B (see page 3-23)

Syntax 4: Class 11A (see page 3-24)

Syntax 4: Class 11B (see page 3-25)

**Example 1**

STL A, DAT11

	Before Instruction		After Instruction
A	FF 8765 4321	A	FF 8765 4321
DP	004	DP	004
Data Memory			
020Bh	1234	020Bh	4321

**Example 2**

STL B, -8, \*AR7-

	Before Instruction		After Instruction
B	FF 8421 1234	B	FF 8421 1234
SXM	0	SXM	0
AR7	0321	AR7	0320
Data Memory			
0321h	0099	0321h	2112

## **STL** *Store Accumulator Low Into Memory*

---

### **Example 3**

STL A, 7, DAT11

Before Instruction		After Instruction	
A	FF 8421 1234	A	FF 8421 1234
DP	004	DP	004
Data Memory			
020Bh	0101	020Bh	1A00

**Syntax** **STLM** *src*, *MMR*

**Operands** **src:** A (accumulator A)  
B (accumulator B)  
**MMR:** Memory-mapped register

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	S	I	A	A	A	A	A	A	A

**Execution** (src(15–0)) → MMR

**Status Bits** None

**Description** This instruction stores the low part of *src* (bits 15–0) into the addressed memory-mapped register *MMR*. The nine MSBs of the effective address are cleared to 0 regardless of the current value of DP or of the upper nine bits of ARx. This instruction allows *src* to be stored in any memory location on data page 0 without modifying the DP field in status register ST0.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 10A (see page 3-22)

**Example 1** STLM A, BRC

	Before Instruction	After Instruction
A	FF 8765 4321	FF 8765 4321
BRC(1Ah)	1234	4321

**Example 2** STLM B, \*AR1-

	Before Instruction	After Instruction
B	FF 8421 1234	FF 8421 1234
AR1	3F17	0016
AR7(17h)	0099	1234

## STM Store Immediate Value Into Memory-Mapped Register

**Syntax** STM #*lk*, *MMR*

**Operands** MMR: Memory-mapped register  
 $-32\,768 \leq lk \leq 32\,767$

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	1	I	A	A	A	A	A	A	A
16-bit constant															

**Execution**  $lk \rightarrow MMR$

**Status Bits** None

**Description** This instruction stores a 16-bit constant *lk* into a memory-mapped register *MMR* or a memory location on data page 0 without modifying the DP field in status register ST0. The nine MSBs of the effective address are cleared to 0 regardless of the current value of DP or of the upper nine bits of ARx.

**Words** 2 words

**Cycles** 2 cycles

**Classes** Class 12A (see page 3-26)

**Example 1** STM 0FFFFh, IMR

	Before Instruction	After Instruction		
IMR	<table border="1"><tr><td>FF01</td></tr></table>	FF01	<table border="1"><tr><td>FFFF</td></tr></table>	FFFF
FF01				
FFFF				

**Example 2** STM 8765h, \*AR7+

	Before Instruction	After Instruction		
AR0	<table border="1"><tr><td>0000</td></tr></table>	0000	<table border="1"><tr><td>8765</td></tr></table>	8765
0000				
8765				
AR7	<table border="1"><tr><td>8010</td></tr></table>	8010	<table border="1"><tr><td>8010</td></tr></table>	8010
8010				
8010				



Syntax	ST <i>src</i> , <i>Ymem</i>    ADD <i>Xmem</i> , <i>dst</i>																																
Operands	src, dst:           A (accumulator A) B (accumulator B) Xmem, Ymem:       Dual data-memory operands dst_:                If <i>dst</i> = A, then <i>dst_</i> = B; if <i>dst</i> = B, then <i>dst_</i> = A																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>S</td><td>D</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	0	0	0	S	D	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	1	0	0	0	0	S	D	X	X	X	X	Y	Y	Y	Y																		
Execution	(src) << (ASM – 16) → Ymem (dst_ ) + (Xmem) << 16 → dst																																
Status Bits	Affected by OVM, SXM, and ASM Affects C and OVdst																																
Description	This instruction stores <i>src</i> shifted by (ASM – 16) in data-memory location <i>Ymem</i> . In parallel, this instruction adds the content of <i>dst_</i> to the data-memory operand <i>Xmem</i> shifted left 16 bits, and stores the result in <i>dst</i> . If <i>src</i> is equal to <i>dst</i> , the value stored in <i>Ymem</i> is the value of <i>src</i> before the execution.																																
Words	1 word																																
Cycles	1 cycle																																
Classes	Class 14 (see page 3-30)																																
Example	ST A, *AR3    ADD *AR5+0%, B																																

	Before Instruction	After Instruction
A	FF 8421 1000	FF 8021 1000
B	00 0000 1111	FF 0422 1000
OVM	0	0
SXM	1	1
ASM	1	1
AR0	0002	0002
AR3	0200	0201
AR5	0300	0302
Data Memory		
0200h	0101	0842
0300h	8001	8001

## ST||LD Store Accumulator With Parallel Load

Syntax	1: ST src, Ymem    LD Xmem, dst 2: ST src, Ymem    LD Xmem, T																																																																
Operands	src, dst: A (accumulator A) B (accumulator B) Xmem, Ymem: Dual data-memory operands																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>S</td><td>D</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>S</td><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	0	1	0	S	D	X	X	X	X	Y	Y	Y	Y	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	0	0	1	S	0	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
1	1	0	0	1	0	S	D	X	X	X	X	Y	Y	Y	Y																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																		
1	1	1	0	0	1	S	0	X	X	X	X	Y	Y	Y	Y																																																		
Execution	1. (src) << (ASM – 16) → Ymem (Xmem) << 16 → dst 2. (src) << (ASM – 16) → Ymem (Xmem) → T																																																																
Status Bits	Affected by OVM and ASM Affects C																																																																
Description	This instruction stores src shifted by (ASM – 16) in data-memory location Ymem. In parallel, this instruction loads the 16-bit dual data-memory operand Xmem to dst or T. If src is equal to dst, the value stored in Ymem is the value of src before the execution.																																																																
Words	1 word																																																																
Cycles	1 cycle																																																																
Classes	Class 14 (see page 3-30)																																																																

**Example 1**

```
ST B, *AR2-
||LD *AR4+, A
```

Before Instruction		After Instruction	
A	00 0000 001C	A	FF 8001 0000
B	FF 8421 1234	B	FF 8421 1234
SXM	1	SXM	1
ASM	1C	ASM	1C
AR2	01FF	AR2	01FE
AR4	0200	AR4	0201
Data Memory			
01FFh	xxxx	01FFh	F842
0200h	8001	0200h	8001

**Example 2**

```
ST A, *AR3
||LD *AR4, T
```

Before Instruction		After Instruction	
A	FF 8421 1234	A	FF 8421 1234
T	3456	T	80FF
ASM	1	ASM	1
AR3	0200	AR3	0200
AR4	0100	AR4	0100
Data Memory			
0200h	0001	0200h	0842
0100h	80FF	0100h	80FF

**ST||MAC[R]** *Store Accumulator With Parallel Multiply Accumulate With/Without Rounding*

<b>Syntax</b>	<b>ST</b> <i>src</i> , <i>Ymem</i> <b>   MAC[R]</b> <i>Xmem</i> , <i>dst</i>																																
<b>Operands</b>	<i>src</i> , <i>dst</i> :           A (accumulator A) B (accumulator B) <i>Xmem</i> , <i>Ymem</i> :   Dual data-memory operands																																
<b>Opcode</b>	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>R</td><td>S</td><td>D</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	1	0	R	S	D	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	1	0	1	0	R	S	D	X	X	X	X	Y	Y	Y	Y																		
<b>Execution</b>	$(src(31-16)) \ll (ASM - 16) \rightarrow Ymem$ If (Rounding) Then Round $((Xmem) \times (T) + (dst)) \rightarrow \times dst$ Else $(Xmem) \times (T) + (dst) \rightarrow dst$																																
<b>Status Bits</b>	Affected by OVM, SXM, ASM, and FRCT Affects C and OVdst																																
<b>Description</b>	<p>This instruction stores the high part of <i>src</i> (bits 31–16) shifted by <math>(ASM - 16)</math> in data-memory location <i>Ymem</i>. In parallel, this instruction multiplies the content of T by the data-memory operand <i>Xmem</i>, adds the value in <i>dst</i> (with or without rounding), and stores the result in <i>dst</i>. If <i>src</i> is equal to <i>dst</i>, the value stored in <i>Ymem</i> is the value of <i>src</i> before the execution of this instruction.</p> <p>If you use the R suffix, this instruction rounds the result of the multiply accumulate operation by adding <math>2^{15}</math> to the result and clearing the LSBs (bits 15–0) to 0.</p>																																
<b>Words</b>	1 word																																
<b>Cycles</b>	1 cycle																																
<b>Classes</b>	Class 14 (see page 3-30)																																

**Example 1**

ST A, \*AR4-  
||MAC \*AR5, B

	Before Instruction	After Instruction
A	00 0011 1111	00 0011 1111
B	00 0000 1111	00 010C 9511
T	0400	0400
ASM	5	5
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
100h	1234	0222
200h	4321	4321

**Example 2**

ST A, \*AR4+  
||MACR \*AR5+, B

	Before Instruction	After Instruction
A	00 0011 1111	00 0011 1111
B	00 0000 1111	00 010D 0000
T	0400	0400
ASM	1C	1C
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
100h	1234	0002
200h	4321	4321

**ST||MAS[R]** *Store Accumulator With Parallel Multiply Subtract With/Without Rounding*

<b>Syntax</b>	<b>ST</b> <i>src</i> , <i>Ymem</i> <b>   MAS[R]</b> <i>Xmem</i> , <i>dst</i>																																
<b>Operands</b>	<i>src</i> , <i>dst</i> :           A (accumulator A) B (accumulator B) <i>Xmem</i> , <i>Ymem</i> :   Dual data-memory operands																																
<b>Opcode</b>	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>R</td><td>S</td><td>D</td><td>X</td><td>X</td><td>X</td><td>X</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	0	1	1	R	S	D	X	X	X	X	Y	Y	Y	Y
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
1	1	0	1	1	R	S	D	X	X	X	X	Y	Y	Y	Y																		
<b>Execution</b>	$(src(31-16)) \ll (ASM - 16) \rightarrow Ymem$ If (Rounding) Then Round $((dst) - (Xmem) \times (T)) \rightarrow dst$ Else $(dst) - (Xmem) \times (T) \rightarrow dst$																																
<b>Status Bits</b>	Affected by OVM, SXM, ASM, and FRCT Affects C and OVdst																																
<b>Description</b>	<p>This instruction stores the high part of <i>src</i> (bits 31–16) shifted by (ASM – 16) in data-memory location <i>Ymem</i>. In parallel, this instruction multiplies the content of T by the data-memory operand <i>Xmem</i>, subtracts the value from <i>dst</i> (with or without rounding), and stores the result in <i>dst</i>. If <i>src</i> is equal to <i>dst</i>, the value stored in <i>Ymem</i> is the value of <i>src</i> before the execution of this instruction.</p> <p>If you use the R suffix, this instruction optionally rounds the result of the multiply subtract operation by adding 2<sup>15</sup> to the result and clearing the LSBs (bits 15–0) to 0.</p>																																
<b>Words</b>	1 word																																
<b>Cycles</b>	1 cycle																																
<b>Classes</b>	Class 14 (see page 3-30)																																

**Example 1**

ST A, \*AR4+  
||MAS \*AR5, B

	Before Instruction	After Instruction
A	00 0011 1111	00 0011 1111
B	00 0000 1111	FF FEF3 8D11
T	0400	0400
ASM	5	5
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
0100h	1234	0222
0200h	4321	4321

**Example 2**

ST A, \*AR4+  
||MASR \*AR5+, B

	Before Instruction	After Instruction
A	00 0011 1111	00 0011 1111
B	00 0000 1111	FF FEF4 0000
T	0400	0400
ASM	0011	0001
FRCT	0	0
AR4	0100	0101
AR5	0200	0201
Data Memory		
0100h	1234	0022
0200h	4321	4321

## STIMPY Store Accumulator With Parallel Multiply

**Syntax**            ST *src*, Ymem  
                     || MPY Xmem, dst

<b>Operands</b>	src, dst:	A (accumulator A) B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	1	1	S	D	X	X	X	X	Y	Y	Y	Y

**Execution**  $(src(31-16)) \ll (ASM - 16) \rightarrow Ymem$   
 $(T) \times (Xmem) \rightarrow dst$

**Status Bits** Affected by OVM, SXM, ASM, and FRCT  
Affects C and OVdst

<b>Description</b>	This instruction stores the high part of <i>src</i> (bits 31–16) shifted by (ASM – 16) in data-memory location <i>Ymem</i> . In parallel, this instruction multiplies the content of T by the 16-bit dual data-memory operand <i>Xmem</i> , and stores the result in <i>dst</i> . If <i>src</i> is equal to <i>dst</i> , then the value stored in <i>Ymem</i> is the value of <i>src</i> before the execution.
--------------------	--

**Words** 1 word

**Cycles** 1 cycle

**Classes**                      Class 14 (see page 3-30)

```
Example      ST A, *AR3+
             ||MPY *AR5+, B
```

Before Instruction		After Instruction	
A	FF 8421 1234	A	FF 8421 1234
B	xx xxxx xxxx	B	00 2000 0000
T	4000	T	4000
ASM	00	ASM	00
FRCT	1	FRCT	1
AR3	0200	AR3	0201
AR5	0300	AR5	0301

Data Memory	
0200h	1111
0300h	4000



**Syntax**                ST *src*, Y*mem*  
                         || SUB X*mem*, *dst*

<b>Operands</b>	src, dst:	A (accumulator A)
		B (accumulator B)
	Xmem, Ymem:	Dual data-memory operands
	dst_:	If <i>dst</i> = A, then <i>dst_</i> = B; if <i>dst</i> = B, then <i>dst_</i> = A.

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	1	0	0	0	1	S	D	X	X	X	X	Y	Y	Y	Y

**Execution**            (src(31-16)) << (ASM - 16) → Ymem  
                             (Xmem) << 16 - (dst\_) → dst

**Status Bits** Affected by OVM, SXM, and ASM  
Affects C and OVDst

<b>Description</b>	This instruction stores the high part of <i>src</i> (bits 31–16) shifted by (ASM – 16) in data-memory location <i>Ymem</i> . In parallel, this instruction subtracts the content of <i>dst_</i> from the 16-bit dual data-memory operand <i>Xmem</i> shifted left 16 bits, and stores the result in <i>dst</i> . If <i>src</i> is equal to <i>dst</i> , then the value stored in <i>Ymem</i> is the value of <i>src</i> before the execution.
--------------------	---

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 14 (see page 3-30)

**Example**            ST A, \*AR3-  
                      | | SUB \*AR5+0<sup>5</sup>, B

	Before Instruction	After Instruction
A	FF 8421 0000	FF 8421 0000
B	00 1000 0001	FF FBEO 0000
ASM	01	01
SXM	1	1
AR0	0002	0002
AR3	01FF	01FE
AR5	0300	0302

Data Memory	
01FFh	1111
0300h	8001

## STRCD *Store T Conditionally*

**Syntax** STRCD *Xmem, cond*

**Operands** Xmem: Dual data-memory operand

The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
AEQ	(A) = 0	0101	BEQ	(B) = 0	1101
ANEQ	(A) $\neq$ 0	0100	BNEQ	(B) $\neq$ 0	1100
AGT	(A) > 0	0110	BGT	(B) > 0	1110
AGEQ	(A) $\geq$ 0	0010	BGEQ	(B) $\geq$ 0	1010
ALT	(A) < 0	0011	BLT	(B) < 0	1011
ALEQ	(A) $\leq$ 0	0111	BLEQ	(B) $\leq$ 0	1111

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	0	1	1	1	0	0	X	X	X	X	C	O	N	D

**Execution** If (cond)  
(T)  $\rightarrow$  Xmem  
Else  
(Xmem)  $\rightarrow$  Xmem

**Status Bits** None

**Description** If the condition is true, this instruction stores the content of T into the data-memory location *Xmem*. If the condition is false, the instruction reads *Xmem* and writes the value in *Xmem* back to the same address; thus, *Xmem* remains the same. Regardless of the condition, *Xmem* is always read and updated.

**Words** 1 word

**Cycles** 1 cycle

**Classes** Class 15 (see page 3-32)

**Example** STRCD \*AR5-, AGT

Before Instruction		After Instruction	
A	00 70FF FFFF	A	00 70FF FFFF
T	4321	T	4321
AR5	0202	AR5	0201
Data Memory			
0202h	1234	0202h	4321

**Syntax**

- 1: **SUB** *Smem*, *src*
- 2: **SUB** *Smem*, **TS**, *src*
- 3: **SUB** *Smem*, **16**, *src* [, *dst*]
- 4: **SUB** *Smem* [, *SHIFT*], *src* [, *dst*]
- 5: **SUB** *Xmem*, *SHFT*, *src*
- 6: **SUB** *Xmem*, *Ymem*, *dst*
- 7: **SUB** *#lk* [, *SHFT*], *src* [, *dst*]
- 8: **SUB** *#lk*, **16**, *src* [, *dst*]
- 9: **SUB** *src* [, *SHIFT*], [, *dst*]
- 10: **SUB** *src*, **ASM** [, *dst*]

**Operands**

*src*, *dst*:     A (accumulator A)  
                   B (accumulator B)  
*Smem*:         Single data-memory operand  
*Xmem*, *Ymem*:   Dual data-memory operands  
 $-32\,768 \leq lk \leq 32\,767$   
 $0 \leq SHFT \leq 15$   
 $-16 \leq SHIFT \leq 15$

**Opcode**

- 1:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 0  | 0  | 0  | 1  | 0  | 0 | S | I | A | A | A | A | A | A | A |
- 2:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 0  | 0  | 0  | 1  | 1  | 0 | S | I | A | A | A | A | A | A | A |
- 3:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 1  | 0  | 0  | 0  | 0  | S | D | I | A | A | A | A | A | A | A |
- 4:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 1  | 1  | 0  | 1  | 1  | 1 | 1 | I | A | A | A | A | A | A | A |
| 0  | 0  | 0  | 0  | 1  | 1  | S | D | 0 | 0 | 1 | S | H | I | F | T |
- 5:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 0  | 0  | 1  | 0  | 0  | 1 | S | X | X | X | X | S | H | F | T |
- 6:
- |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1  | 0  | 1  | 0  | 0  | 0  | 1 | D | X | X | X | X | Y | Y | Y | Y |
- 7:
- |                 |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |
|-----------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15              | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1               | 1  | 1  | 1  | 0  | 0  | S | D | 0 | 0 | 0 | 1 | S | H | F | T |
| 16-bit constant |    |    |    |    |    |   |   |   |   |   |   |   |   |   |   |

## SUB Subtract From Accumulator

8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	S	D	0	1	1	0	0	0	0	1
16-bit constant															

9:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	0	0	1	S	H	I	F	T

10:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	S	D	1	0	0	0	0	0	0	1

### Execution

- 1: (src) – (Smem) → src
- 2: (src) – (Smem) << TS → src
- 3: (src) – (Smem) << 16 → dst
- 4: (src) – (Smem) << SHIFT → dst
- 5: (src) – (Xmem) << SHFT → src
- 6: (Xmem) << 16 – (Ymem) << 16 → dst
- 7: (src) – lk << SHFT → dst
- 8: (src) – lk << 16 → dst
- 9: (src) – (src) << SHIFT → dst
- 10: (src) – (src) << ASM → dst

### Status Bits

Affected by SXM and OVM  
Affects C and OVDst (or OVsrc, if dst = src)

For instruction syntax 3, if the result of the subtraction generates a borrow, the carry bit, C, is cleared to 0; otherwise, C is not affected.

### Description

This instruction subtracts a 16-bit value from the content of the selected accumulator or from the 16-bit operand *Xmem* in dual data-memory addressing mode. The 16-bit value to be subtracted is one of the following:

- ☐ The content of a single data-memory operand (*Smem*)
- ☐ The content of a dual data-memory operand (*Ymem*)
- ☐ A 16-bit immediate operand (*#lk*)
- ☐ The shifted value in *src*

If a *dst* is specified, this instruction stores the result in *dst*. If no *dst* is specified, this instruction stores the result in *src*. Most of the second operands can be shifted. For a left shift:

- ☐ Low-order bits are cleared
- ☐ High-order bits are:
  - Sign extended if SXM = 1
  - Cleared if SXM = 0

For a right shift, the high-order bits are:

- Sign extended if SXM = 1
- Cleared if SXM = 0

---

**Notes:**

The following syntaxes are assembled as a different syntax in certain cases.

- ☐ Syntax 4: If *dst* = *src* and *SHIFT* = 0, then the instruction opcode is assembled as syntax 1.
  - ☐ Syntax 4: If *dst* = *src*, *SHIFT* ≤ 15, and *Smem* indirect addressing mode is included in *Xmem*, then the instruction opcode is assembled as syntax 1.
- 

**Words**

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 word  
 Syntaxes 4, 7, and 8: 2 words

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles**

Syntaxes 1, 2, 3, 5, 6, 9, and 10: 1 cycle  
 Syntaxes 4, 7, and 8: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes**

Syntaxes 1, 2, 3, and 5: Class 3A (see page 3-5)  
 Syntaxes 1, 2, and 3: Class 3B (see page 3-6)  
 Syntax 4: Class 4A (see page 3-7)  
 Syntax 4: Class 4B (see page 3-8)  
 Syntax 6: Class 7 (see page 3-12)  
 Syntaxes 7 and 8: Class 2 (see page 3-4)  
 Syntaxes 9 and 10: Class 1 (see page 3-3)

**Example 1**

SUB \*AR1+, 14, A

	Before Instruction		After Instruction
A	00 0000 1200	A	FF FAC0 1200
C	x	C	0
SXM	1	SXM	1
AR1	0100	AR1	0101
Data Memory			
0100h	1500	0100h	1500

## **SUB** *Subtract From Accumulator*

---

### **Example 2**

SUB A, -8, B

Before Instruction	
A	00 0000 1200
B	00 0000 1800
C	x
SXM	1

After Instruction	
A	00 0000 1200
B	00 0000 17EE
C	0
SXM	1

### **Example 3**

SUB #12345, 8, A, B

Before Instruction	
A	00 0000 1200
B	00 0000 1800
C	x
SXM	1

After Instruction	
A	00 0000 1200
B	FF FFCF D900
C	0
SXM	1

**Syntax** **SUBB** *Smem*, *src*

**Operands** *src*: A (accumulator A)  
 B (accumulator B)  
*Smem*: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	D	I	A	A	A	A	A	A	A

**Execution** (*src*) – (*Smem*) – (logical inversion of C) → *src*

**Status Bits** Affected by OVM and C  
 Affects C and OVsrc

**Description** This instruction subtracts the content of the 16-bit single data-memory operand *Smem* and the logical inverse of the carry bit, C, from *src* without sign extension.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an *Smem*.

**Classes** Class 3A (see page 3-5)  
 Class 3B (see page 3-6)

**Example 1** SUBB DAT5, A

	Before Instruction	After Instruction
A	00 0000 0006	FF FFFF FFFF
C	0	0
DP	008	008
Data Memory		
0405h	0006	0006

**Example 2**

SUBB \*AR1+, B

	Before Instruction	After Instruction
B	FF 8000 0006	FF 8000 0000
C	1	1
OVM	1	1
AR1	0405	0406
Data Memory		
0405h	0006	0006

## SUBC *Subtract Conditionally*

<b>Syntax</b>	<b>SUBC</b> <i>Smem</i> , <i>src</i>																																
<b>Operands</b>	<i>Smem</i> : Single data-memory operand <i>src</i> : A (accumulator A) B (accumulator B)																																
<b>Opcode</b>	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	1	1	1	1	S	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	0	0	1	1	1	1	S	I	A	A	A	A	A	A	A																		
<b>Execution</b>	$(src) - ((Smem) \ll 15) \rightarrow \text{ALU output}$ If ALU output $\geq 0$ Then $((\text{ALU output}) \ll 1) + 1 \rightarrow src$ Else $(src) \ll 1 \rightarrow src$																																
<b>Status Bits</b>	Affected by SXM Affects C and OVsrc																																
<b>Description</b>	<p>This instruction subtracts the 16-bit single data-memory operand <i>Smem</i>, left-shifted 15 bits, from the content of <i>src</i>. If the result is greater than 0, it is shifted 1 bit left, 1 is added to the result, and the result is stored in <i>src</i>. Otherwise, this instruction shifts the content of <i>src</i> 1 bit left and stores the result in <i>src</i>.</p> <p>The divisor and the dividend are both assumed to be positive in this instruction. The SXM bit affects this operation in these ways:</p> <ul style="list-style-type: none"><li><input type="checkbox"/> If SXM = 1, the divisor must have a 0 value in the MSB.</li><li><input type="checkbox"/> If SXM = 0, any 16-bit divisor value produces the expected results.</li></ul> <p>The dividend, which is in <i>src</i>, must initially be positive (bit 31 must be 0) and must remain positive following the accumulator shift, which occurs in the first portion of the instruction.</p> <p>This instruction affects OVA or OVB (depending on <i>src</i>) but is not affected by OVM; therefore, <i>src</i> does not saturate on positive or negative overflows when executing this instruction.</p>																																
<b>Words</b>	1 word  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
<b>Cycles</b>	1 cycle  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																



**Classes**

Class 3A (see page 3-5)

Class 3B (see page 3-6)

**Example 1**

SUBC DAT2, A

Before Instruction		After Instruction	
A	00 0000 0004	A	00 0000 0008
C	x	C	0
DP	006	DP	006
Data Memory			
0302h	0001	0302h	0001

**Example 2**

RPT #15

SUBC \*AR1, B

Before Instruction		After Instruction	
B	00 0000 0041	B	00 0002 0009
C	x	C	1
AR1	1000	AR1	1000
Data Memory			
1000h	0007	1000h	0007

## SUBS *Subtract From Accumulator With Sign Extension Suppressed*

**Syntax** SUBS *Smem*, *src*

**Operands** Smem: Single data-memory operand  
src: A (accumulator A)  
B (accumulator B)

Opcode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	1	0	1	S	I	A	A	A	A	A	A	A

**Execution** unsigned(src) – (Smem) → src

**Status Bits** Affected by OVM  
Affects C and OVsrc

**Description** This instruction subtracts the content of the 16-bit single data-memory operand *Smem* from the content of *src*. *Smem* is considered a 16-bit unsigned number regardless of the value of SXM. The result is stored in *src*.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 1 cycle

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 3A (see page 3-5)  
Class 3B (see page 3-6)

**Example** SUBS \*AR2-, B

	Before Instruction	After Instruction
B	00 0000 0002	FF FFFF 0FFC
C	x	0
AR2	0100	00FF
Data Memory		
0100h	F006	F006

**Syntax** **TRAP K****Operands**  $0 \leq K \leq 31$ 

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	1	1	0	K	K	K	K	K

**Execution**

(SP) - 1 → SP  
(PC) + 1 → TOS  
Interrupt vector specified by K → PC

**Status Bits** None

**Description**

This instruction transfers program control to the interrupt vector specified by K. This instruction allows you to use your software to execute any interrupt service routine. For a list of interrupts and their corresponding K value, see Appendix B.

This instruction pushes PC + 1 onto the data-memory location addressed by SP. This enables a return instruction to retrieve the pointer to the instruction after the trap from the data-memory location addressed by SP. This instruction is not maskable and is not affected by INTM nor does it affect INTM.

**Note:**

This instruction is not repeatable.

**Words** 1 word**Cycles** 3 cycles**Classes** Class 35 (see page 3-72)**Example** TRAP 10h

	Before Instruction	After Instruction
PC	1233	FFC0
SP	03FF	03FE
Data Memory		
03FEh	9653	1234

## WRITA Write Data to Program Memory Addressed by Accumulator A

**Syntax** WRITA *Smem*

**Operands** Smem: Single data-memory operand

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1	A	A	A	A	A	A	A

**Execution**

A → PAR  
If (RC) ≠ 0  
Then  
    (Smem) → (Pmem addressed by PAR)  
    (PAR) + 1 → PAR  
    (RC) – 1 → RC  
Else  
    (Smem) → (Pmem addressed by PAR)

**Status Bits** None

**Description** This instruction transfers a word from a data-memory location specified by *Smem* to a program-memory location. The program-memory address is defined by accumulator A as follows, depending on the specific device.

'541–'546	'548
A(15–0)	A(22–0)

This instruction can be used with the repeat instruction to move consecutive words (using indirect addressing) in data memory to a continuous program-memory space addressed by PAR by automatically incrementing PAR. The initial value is set with the 16 LSBs of accumulator A. The source and destination blocks in memory do not have to be entirely on-chip or off-chip. When used with repeat, this instruction becomes a single-cycle instruction once the repeat pipeline is started.

The content of accumulator A is not affected by this instruction.

**Words** 1 word

Add 1 word when using long-offset indirect addressing or absolute addressing with an Smem.

**Cycles** 5 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

**Classes** Class 26A (see page 3-60)  
Class 26B (see page 3-62)

**Example**

WRITA DAT5

	Before Instruction	After Instruction
A	00 0000 0257	00 0000 0257
DP	032	032
Program Memory		
0257h	0306	4339
Data Memory		
1905h	4339	4339

**XC** *Execute Conditionally***Syntax** **XC** *n, cond* [, *cond* [, *cond* ]]**Operands** *n* = 1 or 2The following table lists the conditions (*cond* operand) for this instruction.

Cond	Description	Condition Code	Cond	Description	Condition Code
BIO	$\overline{BIO}$ low	0000 0011	NBIO	$\overline{BIO}$ high	0000 0010
C	$C = 1$	0000 1100	NC	$C = 0$	0000 1000
TC	$TC = 1$	0011 0100	NTC	$TC = 0$	0010 0000
AEQ	$(A) = 0$	0100 0101	BEQ	$(B) = 0$	0100 1101
ANEQ	$(A) \neq 0$	0100 0100	BNEQ	$(B) \neq 0$	0100 1100
AGT	$(A) > 0$	0100 0110	BGT	$(B) > 0$	0100 1110
AGEQ	$(A) \geq 0$	0100 0010	BGEQ	$(B) \geq 0$	0100 1010
ALT	$(A) < 0$	0100 0011	BLT	$(B) < 0$	0100 1011
ALEQ	$(A) \leq 0$	0100 0111	BLEQ	$(B) \leq 0$	0100 1111
AOV	A overflow	0111 0000	BOV	B overflow	0111 1000
ANOV	A no overflow	0110 0000	BNOV	B no overflow	0110 1000
UNC	Unconditional	0000 0000			

**Opcode**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	N	1	C	C	C	C	C	C	C	C

Syntax <i>n</i>	Opcode <i>N</i>
1	0
2	1

**Execution**      If (*cond*)  
                      Then  
                      Next *n* instructions are executed  
                      Else  
                      Execute NOP for next *n* instructions

**Status Bits**      None

**Description**

The execution of this instruction depends on the value of *n* and the selected conditions:

- ☐ If *n* = 1 and the condition(s) is met, the 1-word instruction following this instruction is executed.
- ☐ If *n* = 2 and the condition(s) is met, the one 2-word instruction or the two 1-word instructions following this instruction are executed.
- ☐ If the condition(s) is not met, one or two nops are executed depending on the value of *n*.

This instruction tests multiple conditions before executing and can test the conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

**Group 1:** You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time.

**Group 2:** You can select up to three conditions. Each of these conditions must be from a different category (category A, B, or C); you cannot have two conditions from the same category. For example, you can test TC, C, and BIO at the same time but you cannot test NTC, C, and NC at the same time.

Conditions for This Instruction				
Group 1		Group 2		
Category A	Category B	Category A	Category B	Category C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

This instruction and the two instruction words following this instruction are uninterruptible.

## **XC** *Execute Conditionally*

### **Note:**

The conditions tested are sampled two full cycles before this instruction is executed. Therefore, if the two 1-word instructions or one 2-word instruction modifies the conditions, there is no effect on the execution of this instruction, but if the conditions are modified during the two slots, the interrupt operation using this instruction can cause undesirable results.

This instruction is not repeatable.

<b>Words</b>	1 word
<b>Cycles</b>	1 cycle
<b>Classes</b>	Class 1 (see page 3-3)
<b>Example</b>	<pre>XC 1, ALEQ MAR *AR1+ ADD A, DAT100</pre>

Before Instruction		After Instruction	
A	FF FFFF FFFF	A	FF FFFF FFFF
AR1	0032	AR1	0033

If the content of accumulator A is less than or equal to 0, AR1 is modified before the execution of the addition instruction.



Syntax	1: <b>XOR</b> <i>Smem</i> , <i>src</i> 2: <b>XOR</b> <i>#lk</i> [, <i>SHFT</i> ], <i>src</i> [, <i>dst</i> ] 3: <b>XOR</b> <i>#lk</i> , 16, <i>src</i> [, <i>dst</i> ] 4: <b>XOR</b> <i>src</i> [, <i>SHIFT</i> ] [, <i>dst</i> ]																																																																																																																																																																
Operands	src, dst: A (accumulator A) B (accumulator B) Smem: Single data-memory operand 0 ≤ SHFT ≤ 15 −16 ≤ SHIFT ≤ 15 0 ≤ lk ≤ 65 535																																																																																																																																																																
Opcode	1: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>S</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table> 2: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>D</td><td>0</td><td>1</td><td>0</td><td>1</td><td>S</td><td>H</td><td>F</td><td>T</td></tr><tr><td colspan="16">16-bit constant</td></tr></table> 3: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>D</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td colspan="16">16-bit constant</td></tr></table> 4: <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>S</td><td>D</td><td>1</td><td>1</td><td>0</td><td>S</td><td>H</td><td>I</td><td>F</td><td>T</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	1	1	1	0	S	I	A	A	A	A	A	A	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	S	D	0	1	0	1	S	H	F	T	16-bit constant																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	S	D	0	1	1	0	0	1	0	1	16-bit constant																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1	1	1	0	0	S	D	1	1	0	S	H	I	F	T
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
0	0	0	1	1	1	0	S	I	A	A	A	A	A	A	A																																																																																																																																																		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
1	1	1	1	0	0	S	D	0	1	0	1	S	H	F	T																																																																																																																																																		
16-bit constant																																																																																																																																																																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
1	1	1	1	0	0	S	D	0	1	1	0	0	1	0	1																																																																																																																																																		
16-bit constant																																																																																																																																																																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																		
1	1	1	1	0	0	S	D	1	1	0	S	H	I	F	T																																																																																																																																																		
Execution	1: (Smem) XOR (src) → src 2: lk << SHFT XOR (src) → dst 3: lk << 16 XOR (src) → dst 4: (src) << SHIFT XOR (dst) → dst																																																																																																																																																																
Status Bits	None																																																																																																																																																																
Description	This instruction executes an exclusive OR of the 16-bit single data-memory operand <i>Smem</i> (shifted as indicated in the instruction) with the content of the selected accumulator and stores the result in <i>dst</i> or <i>src</i> , as specified. For a left shift, the low-order bits are cleared and the high-order bits are not sign extended. For a right shift, the sign is not extended.																																																																																																																																																																
Words	Syntaxes 1 and 4: 1 word Syntaxes 2 and 3: 2 words  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																																																																																																																																																

## **XOR** *Exclusive OR With Accumulator*

### **Cycles**

Syntaxes 1 and 4: 1 cycle

Syntaxes 2 and 3: 2 cycles

Add 1 cycle when using long-offset indirect addressing or absolute addressing with an Smem.

### **Classes**

Syntax 1: Class 3A (see page 3-5)

Syntax 1: Class 3B (see page 3-6)

Syntaxes 2 and 3: Class 2 (see page 3-4)

Syntax 4: Class 1 (see page 3-3)

### **Example 1**

XOR \*AR3+, A

	Before Instruction	After Instruction
A	00 00FF 1200	00 00FF 0700
AR3	0100	0101
Data Memory		
0100h	1500	1500

### **Example 2**

XOR A, +3, B

	Before Instruction	After Instruction
A	00 0000 1200	00 0000 1200
B	00 0000 1800	00 0000 8800

Syntax	XORM #lk, Smem																																
Operands	Smem: Single data-memory operand 0 ≤ lk ≤ 65 535																																
Opcode	<table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>I</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td><td>A</td></tr></table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	1	1	0	1	0	1	0	I	A	A	A	A	A	A	A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	1	1	0	1	0	1	0	I	A	A	A	A	A	A	A																		
Execution	lk XOR (Smem) → Smem																																
Status Bits	None																																
Description	This instruction executes an exclusive OR of the content of a data-memory location <i>Smem</i> with a 16-bit constant <i>lk</i> . The result is written to <i>Smem</i> .																																
<div><b>Note:</b> This instruction is not repeatable.</div>																																	
Words	2 words  Add 1 word when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Cycles	2 cycles  Add 1 cycle when using long-offset indirect addressing or absolute addressing with an <i>Smem</i> .																																
Classes	Class 18A (see page 3-39) Class 18B (see page 3-39)																																
Example	XORM 0404h, *AR4-																																

	Before Instruction	After Instruction
AR4	0100	00FF
Data Memory		
0100h	4444	4040

# Condition Codes

---

---

This appendix lists the conditions for conditional instructions (Table A–1) and the combination of conditions that can be tested (Table A–2). Conditional instructions can test conditions individually or in combination with other conditions. You can combine conditions from only one group as follows:

- Group 1: You can select up to two conditions. Each of these conditions must be from a different category (category A or B); you cannot have two conditions from the same category. For example, you can test EQ and OV at the same time but you cannot test GT and NEQ at the same time.
- Group 2: You can select up to three conditions. Each of these conditions must be from a different category (category A, B, or C); you cannot have two conditions from the same category. For example, you can test TC, C, and BIO at the same time but you cannot test NTC, C, and NC at the same time.

## Conditions for Conditional Instructions

Table A-1. Conditions for Conditional Instructions

Operand	Condition	Description
AEQ	$A = 0$	Accumulator A equal to 0
BEQ	$B = 0$	Accumulator B equal to 0
ANEQ	$A \neq 0$	Accumulator A not equal to 0
BNEQ	$B \neq 0$	Accumulator B not equal to 0
ALT	$A < 0$	Accumulator A less than 0
BLT	$B < 0$	Accumulator B less than 0
ALEQ	$A \leq 0$	Accumulator A less than or equal to 0
BLEQ	$B \leq 0$	Accumulator B less than or equal to 0
AGT	$A > 0$	Accumulator A greater than 0
BGT	$B > 0$	Accumulator B greater than 0
AGEQ	$A \geq 0$	Accumulator A greater than or equal to 0
BGEQ	$B \geq 0$	Accumulator B greater than or equal to 0
AOV†	$AOV = 1$	Accumulator A overflow detected
BOV†	$BOV = 1$	Accumulator B overflow detected
ANOV†	$AOV = 0$	No accumulator A overflow detected
BNOV†	$BOV = 0$	No accumulator B overflow detected
C†	$C = 1$	ALU carry set to 1
NC†	$C = 0$	ALU carry clear to 0
TC†	$TC = 1$	Test/Control flag set to 1
NTC†	$TC = 0$	Test/Control flag cleared to 0
BIO†	$\overline{BIO}$ low	$\overline{BIO}$ signal is low
NBIO†	$\overline{BIO}$ high	$\overline{BIO}$ signal is high
UNC†	none	Unconditional operation

† Cannot be used with conditional store instructions

Table A-2. Groupings of Conditions

Group 1		Group 2		
Category A	Category B	Category A	Category B	Category C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

## **Interrupt Locations and Priority Tables**

---

---

This appendix lists the '54x interrupt locations and priorities for each individual device type.

Table B-1. '541 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location	Function
0	1	$\overline{RS}/SINTR$	0	Reset (hardware and software reset)
1	2	$\overline{NMI}/SINT16$	4	Nonmaskable interrupt
2	–	SINT17	8	Software interrupt #17
3	–	SINT18	C	Software interrupt #18
4	–	SINT19	10	Software interrupt #19
5	–	SINT20	14	Software interrupt #20
6	–	SINT21	18	Software interrupt #21
7	–	SINT22	1C	Software interrupt #22
8	–	SINT23	20	Software interrupt #23
9	–	SINT24	24	Software interrupt #24
10	–	SINT25	28	Software interrupt #25
11	–	SINT26	2C	Software interrupt #26
12	–	SINT27	30	Software interrupt #27
13	–	SINT28	34	Software interrupt #28
14	–	SINT29	38	Software interrupt #29; reserved
15	–	SINT30	3C	Software interrupt #30; reserved
16	3	$\overline{INT0}/SINT0$	40	External user interrupt #0
17	4	$\overline{INT1}/SINT1$	44	External user interrupt #1
18	5	$\overline{INT2}/SINT2$	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	RINT0/SINT4	50	Serial port 0 receive interrupt
21	8	XINT0/SINT5	54	Serial port 0 transmit interrupt
22	9	RINT1/SINT6	58	Serial port 1 receive interrupt
23	10	XINT1/SINT7	5C	Serial port 1 transmit interrupt
24	11	$\overline{INT3}/SINT8$	60	External user interrupt #3
25–31	–		64–7F	Reserved



Table B-2. '542 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location	Function
0	1	$\overline{RS}/SINTR$	0	Reset (hardware and software reset)
1	2	$\overline{NMI}/SINT16$	4	Nonmaskable interrupt
2	—	SINT17	8	Software interrupt #17
3	—	SINT18	C	Software interrupt #18
4	—	SINT19	10	Software interrupt #19
5	—	SINT20	14	Software interrupt #20
6	—	SINT21	18	Software interrupt #21
7	—	SINT22	1C	Software interrupt #22
8	—	SINT23	20	Software interrupt #23
9	—	SINT24	24	Software interrupt #24
10	—	SINT25	28	Software interrupt #25
11	—	SINT26	2C	Software interrupt #26
12	—	SINT27	30	Software interrupt #27
13	—	SINT28	34	Software interrupt #28
14	—	SINT29	38	Software interrupt #29, reserved
15	—	SINT30	3C	Software interrupt #30, reserved
16	3	$\overline{INT0}/SINT0$	40	External user interrupt #0
17	4	$\overline{INT1}/SINT1$	44	External user interrupt #1
18	5	$\overline{INT2}/SINT2$	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	BRINT0/SINT4	50	Buffered serial port receive interrupt
21	8	BXINT0/SINT5	54	Buffered serial port transmit interrupt
22	9	TRINT/SINT6	58	TDM serial port receive interrupt
23	10	TXINT/SINT7	5C	TDM serial port transmit interrupt
24	11	$\overline{INT3}/SINT8$	60	External user interrupt #3
25	12	$\overline{HPINT}/SINT9$	64	HPI interrupt
26–31	—		68–7F	Reserved

Table B-3. '543 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location	Function
0	1	$\overline{RS}/SINTR$	0	Reset (hardware and software reset)
1	2	$\overline{NMI}/SINT16$	4	Nonmaskable interrupt
2	—	SINT17	8	Software interrupt #17
3	—	SINT18	C	Software interrupt #18
4	—	SINT19	10	Software interrupt #19
5	—	SINT20	14	Software interrupt #20
6	—	SINT21	18	Software interrupt #21
7	—	SINT22	1C	Software interrupt #22
8	—	SINT23	20	Software interrupt #23
9	—	SINT24	24	Software interrupt #24
10	—	SINT25	28	Software interrupt #25
11	—	SINT26	2C	Software interrupt #26
12	—	SINT27	30	Software interrupt #27
13	—	SINT28	34	Software interrupt #28
14	—	SINT29	38	Software interrupt #29, reserved
15	—	SINT30	3C	Software interrupt #30, reserved
16	3	$\overline{INT0}/SINT0$	40	External user interrupt #0
17	4	$\overline{INT1}/SINT1$	44	External user interrupt #1
18	5	$\overline{INT2}/SINT2$	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	BRINT0/SINT4	50	Buffered serial port receive interrupt
21	8	BXINT0/SINT5	54	Buffered serial port transmit interrupt
22	9	TRINT/SINT6	58	TDM serial port receive interrupt
23	10	TXINT/SINT7	5C	TDM serial port transmit interrupt
24	11	$\overline{INT3}/SINT8$	60	External user interrupt #3
25–31	—		64–7F	Reserved

Table B-4. '545 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location	Function
0	1	$\overline{RS}/SINTR$	0	Reset (hardware and software reset)
1	2	$\overline{NMI}/SINT16$	4	Nonmaskable interrupt
2	—	SINT17	8	Software interrupt #17
3	—	SINT18	C	Software interrupt #18
4	—	SINT19	10	Software interrupt #19
5	—	SINT20	14	Software interrupt #20
6	—	SINT21	18	Software interrupt #21
7	—	SINT22	1C	Software interrupt #22
8	—	SINT23	20	Software interrupt #23
9	—	SINT24	24	Software interrupt #24
10	—	SINT25	28	Software interrupt #25
11	—	SINT26	2C	Software interrupt #26
12	—	SINT27	30	Software interrupt #27
13	—	SINT28	34	Software interrupt #28
14	—	SINT29	38	Software interrupt #29, reserved
15	—	SINT30	3C	Software interrupt #30, reserved
16	3	$\overline{INT0}/SINT0$	40	External user interrupt #0
17	4	$\overline{INT1}/SINT1$	44	External user interrupt #1
18	5	$\overline{INT2}/SINT2$	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	BRINT0/SINT4	50	Buffered serial port receive interrupt
21	8	BXINT0/SINT5	54	Buffered serial port transmit interrupt
22	9	RINT1/SINT6	58	Serial port receive interrupt
23	10	XINT1/SINT7	5C	Serial port transmit interrupt
24	11	$\overline{INT3}/SINT8$	60	External user interrupt #3
25	12	$\overline{HPINT}/SINT9$	64	HPI interrupt
26–31	—		68–7F	Reserved

Table B-5. '546 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location	Function
0	1	$\overline{RS}/SINTR$	0	Reset (hardware and software reset)
1	2	$\overline{NMI}/SINT16$	4	Nonmaskable interrupt
2	—	SINT17	8	Software interrupt #17
3	—	SINT18	C	Software interrupt #18
4	—	SINT19	10	Software interrupt #19
5	—	SINT20	14	Software interrupt #20
6	—	SINT21	18	Software interrupt #21
7	—	SINT22	1C	Software interrupt #22
8	—	SINT23	20	Software interrupt #23
9	—	SINT24	24	Software interrupt #24
10	—	SINT25	28	Software interrupt #25
11	—	SINT26	2C	Software interrupt #26
12	—	SINT27	30	Software interrupt #27
13	—	SINT28	34	Software interrupt #28
14	—	SINT29	38	Software interrupt #29, reserved
15	—	SINT30	3C	Software interrupt #30, reserved
16	3	$\overline{INT0}/SINT0$	40	External user interrupt #0
17	4	$\overline{INT1}/SINT1$	44	External user interrupt #1
18	5	$\overline{INT2}/SINT2$	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	BRINT0/SINT4	50	Buffered serial port receive interrupt
21	8	BXINT0/SINT5	54	Buffered serial port transmit interrupt
22	9	RINT1/SINT6	58	Serial port receive interrupt
23	10	XINT1/SINT7	5C	Serial port transmit interrupt
24	11	$\overline{INT3}/SINT8$	60	External user interrupt #3
25-31	—		64-7F	Reserved

Table B-6. '548 Interrupt Locations and Priorities

TRAP/INTR Number (K)	Priority	Name	Location	Function
0	1	$\overline{RS}/SINTR$	0	Reset (hardware and software reset)
1	2	$\overline{NMI}/SINT16$	4	Nonmaskable interrupt
2	—	SINT17	8	Software interrupt #17
3	—	SINT18	C	Software interrupt #18
4	—	SINT19	10	Software interrupt #19
5	—	SINT20	14	Software interrupt #20
6	—	SINT21	18	Software interrupt #21
7	—	SINT22	1C	Software interrupt #22
8	—	SINT23	20	Software interrupt #23
9	—	SINT24	24	Software interrupt #24
10	—	SINT25	28	Software interrupt #25
11	—	SINT26	2C	Software interrupt #26
12	—	SINT27	30	Software interrupt #27
13	—	SINT28	34	Software interrupt #28
14	—	SINT29	38	Software interrupt #29, reserved
15	—	SINT30	3C	Software interrupt #30, reserved
16	3	$\overline{INT0}/SINT0$	40	External user interrupt #0
17	4	$\overline{INT1}/SINT1$	44	External user interrupt #1
18	5	$\overline{INT2}/SINT2$	48	External user interrupt #2
19	6	TINT/SINT3	4C	Internal timer interrupt
20	7	BRINT0/SINT4	50	Buffered serial port 0 receive interrupt
21	8	BXINT0/SINT5	54	Buffered serial port 0 transmit interrupt
22	9	TRINT/SINT6	58	TDM serial port receive interrupt
23	10	TXINT/SINT7	5C	TDM serial port transmit interrupt
24	11	$\overline{INT3}/SINT8$	60	External user interrupt #3
25	12	$\overline{HPINT}/SINT9$	64	HPI interrupt
26	13	BRINT1/SINT10	68	Buffered serial port 1 receive interrupt
27	14	BXINT1/SINT11	6C	Buffered serial port 1 transmit interrupt
28-31	—		70-7F	Reserved

## Interrupt and Status Registers

This appendix shows the bit fields of the '54x interrupt and status registers. The following table defines terms used in identifying these register fields.

*Table C–1. Register Field Terms and Definitions*

Term	Definition
ARP	Auxiliary register pointer
ASM	Accumulator shift mode
AVIS	Address visibility mode
BRAF	Block repeat active flag
BRINT, BRINT1, BRINT0	Buffered serial port receive interrupts
BXINT, BXINT1, BXINT0	Buffered serial port transmit interrupts
C	Carry
CLKOFF	CLOCKOUT off
CMPT	Compatibility mode
CPL	Compiler mode
C16	Dual 16-bit/double-precision arithmetic mode
DP	Data page pointer
DROM	Data ROM
FRCT	Fractional mode
HM	Hold mode
HPINT	HPI interrupt
INTM	Interrupt mask
INT0–INT3	External user interrupts
IPTR	Interrupt vector pointer
MP/ $\overline{MC}$	Microprocessor/microcomputer

*Table C–1. Register Field Terms and Definitions (Continued)*

Term	Definition
OVA	Overflow flag A
OVB	Overflow flag B
OVL	RAM overlay
OVM	Overflow mode
RINT, RINT0, RINT1	Serial port receive interrupts
Resvd	Reserved
SMUL	Saturation on multiplication
SST	Saturation on store
SXM	Sign-extension mode
TC	Test/control flag
TINT	Internal timer interrupt
TRINT	TDM serial port receive interrupt
TXINT	TDM serial port transmit interrupt
XF	External flag status
XINT, XINT0, XINT1	Serial port transmit interrupts

Figure C-1. Interrupt Flag Register (IFR)

(a) '541 IFR

15-12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	XINT1	RINT1	XINT0	RINT0	TINT	INT2	INT1	INT0

(b) '542 IFR

15-12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	HPINT	INT3	TXINT	TRINT	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(c) '543 IFR

15-12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	TXINT	TRINT	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(d) '545 IFR

15-12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	HPINT	INT3	XINT1	RINT1	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(e) '546 IFR

15-12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	XINT1	RINT1	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(f) '548 IFR

15-12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	BXINT1	BRINT1	HPINT	INT3	TXINT	TRINT	BXINT0	BRINT0	TINT	INT2	INT1	INT0



# Interrupt Mask Register (IMR)

Figure C–2. Interrupt Mask Register (IMR)

(a) '541 IMR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	XINT1	RINT1	XINT0	RINT0	TINT	INT2	INT1	INT0

(b) '542 IMR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	HPINT	INT3	TXINT	TRINT	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(c) '543 IMR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	TXINT	TRINT	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(d) '545 IMR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	HPINT	INT3	XINT1	RINT1	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(e) '546 IMR

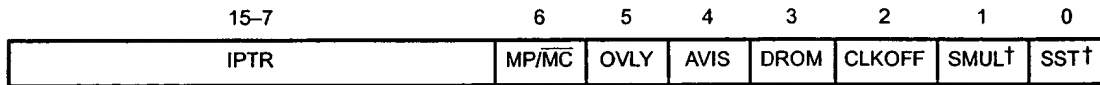
15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	Resvd	Resvd	Resvd	INT3	XINT1	RINT1	BXINT0	BRINT0	TINT	INT2	INT1	INT0

(f) '548 IMR

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Resvd	BXINT1	BRINT1	HPINT	INT3	TXINT	TRINT	BXINT0	BRINT0	TINT	INT2	INT1	INT0

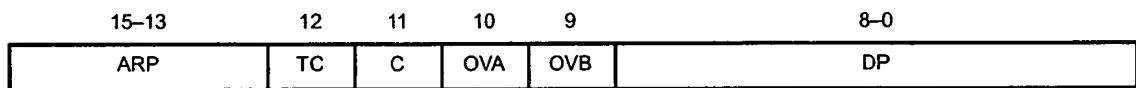
*Processor Mode Status Register (PMST) / Status Register 0 (ST0) / Status Register 1 (ST1)*

*Figure C-3. Processor Mode Status Register (PMST)*

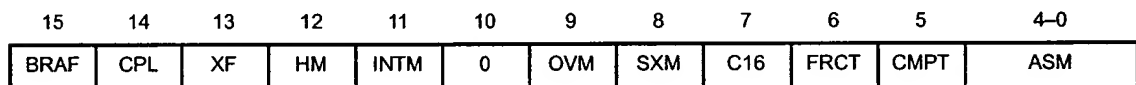


† Only on the LP devices; reserved bits on all other devices

*Figure C-4. Status Register 0 (ST0)*



*Figure C-5. Status Register 1 (ST1)*



# Glossary

## A

**A:** See *accumulator A*.

**accumulator:** A register that stores the results of an operation and provides an input for subsequent arithmetic logic unit (ALU) operations.

**accumulator A:** One of two 40-bit registers that store the result of an operation and provide an input for subsequent arithmetic logic unit (ALU) operations.

**accumulator B:** One of two 40-bit registers that store the result of an operation and provide an input for subsequent arithmetic logic unit (ALU) operations.

**accumulator shift mode bits (ASM):** A 5-bit field in ST1 that specifies a shift value (from –16 to 15) that is used to shift an accumulator value when executing certain instructions, such as instructions with parallel loads and stores.

**address:** The location of a word in memory.

**address visibility mode bit (AVIS):** A bit in PMST that determines whether or not the internal program address appears on the device's external address bus pins.

**addressing mode:** The method by which an instruction calculates the location of an object in memory.

**AG:** *Accumulator guard bits.* An 8-bit register that contains bits 39–32 (the guard bits) of an accumulator. Both accumulator A and accumulator B have guards bits.

**AH:** *Accumulator A high word.* Bits 31–16 of accumulator A.

**AL:** *Accumulator A low word.* Bits 15–0 of accumulator A.

**ALU:** *Arithmetic logic unit.* The part of the CPU that performs arithmetic and logic operations.

**AR0–AR7:** See *auxiliary registers*.

**ARAU:** See *auxiliary register arithmetic unit*.

**ARP:** See *auxiliary register pointer*.

**ASM:** See *accumulator shift mode bits*.

**auxiliary register arithmetic unit (ARAU):** An unsigned, 16-bit arithmetic logic unit (ALU) used to calculate indirect addresses using auxiliary registers.

**auxiliary register file:** The area in data memory containing the eight 16-bit auxiliary registers. See also *auxiliary registers*.

**auxiliary register pointer (ARP):** A 3-bit field in ST0 used as a pointer to the currently-selected auxiliary register, when the device is operating in 'C5x/C2xx compatibility mode.

**auxiliary registers (AR0–AR7):** Eight 16-bit registers that are used as pointers to an address within data space. These registers are operated on by the auxiliary register arithmetic units (ARAUs) and are selected by the auxiliary register pointer (ARP). See also *auxiliary register arithmetic unit*.

**AVIS:** See *address visibility mode bit*.

## B

**B:** See *accumulator B*.

**barrel shifter:** A unit that rotates bits in a word.

**BG:** *Accumulator B guard bits.* An 8-bit register that contains bits 39–32 (the guard bits) of accumulator B.

**BH:** *Accumulator B high word.* Bits 31–16 of accumulator B.

**BL:** *Accumulator B low word.* Bits 15–0 of accumulator B.

**block-repeat active flag (BRAf):** A 1-bit flag in ST1 that indicates whether or not a block repeat is currently active.

**block-repeat counter (BRC):** A 16-bit register that specifies the number of times a block of code is to be repeated when a block repeat is performed.

**block-repeat end address register (REA):** A 16-bit memory-mapped register containing the end address of a code segment being repeated.

**block-repeat start address register (RSA):** A 16-bit memory-mapped register containing the start address of a code segment being repeated.

**boot:** The process of loading a program into program memory.

**boot loader:** A built-in segment of code that transfers code from an external source to program memory at power-up.

**BRC:** See *block-repeat counter*.

**butterfly:** A kernel function for computing an N-point fast Fourier transform (FFT), where N is a power of 2. The combinational pattern of inputs resembles butterfly wings.

## C

**C16:** A bit in ST1 that determines whether the ALU operates in dual 16-bit mode or in double-precision mode.

**CAB:** *C address bus*. A bus that carries addresses needed for accessing data memory.

**carry bit (C):** A bit used by the ALU in extended arithmetic operations and accumulator shifts and rotates. The carry bit can be tested by conditional instructions.

**CB:** *C bus*. A bus that carries operands that are read from data memory.

**CMPT:** See *compatibility mode bit*.

**CNF:** See *configuration control bit*.

**code:** A set of instructions written to perform a task.

**cold boot:** The process of loading a program into program memory at power-up.

**compatibility mode bit (CMPT):** A bit in ST1 that determines whether or not the auxiliary register pointer (ARP) is used to select an auxiliary register in single indirect addressing mode.

**compiler mode bit (CPL):** A bit in ST1 that determines whether the CPU uses the data page pointer or the stack pointer to generate data memory addresses in direct addressing mode.

**configuration control bit (CNF):** A bit in ST1 used to indicate when on-chip RAM is mapped to program or data space.

**CPL:** See *compiler mode bit*.

## D

**DAB:** *D address bus*. A bus that carries addresses needed for accessing data memory.

**DAB address register (DAR):** An address register used to address the data memory operand represented by the immediate value in MVDM and MVKD instructions.

**DAGEN:** See *data address generation logic*.

**DAR:** See *DAB address register*.

**DARAM:** *Dual-access RAM*. Memory that can be read from and written to in the same clock cycle.

**data address bus:** A group of connections used to route data memory addresses. The LMM has three 16-bit buses that carry data memory addresses: *CAB*, *DAB*, and *EAB*.

**data address generation logic (DAGEN):** Logic circuitry that generates the addresses for data memory reads and writes. See also *program address generation logic*.

**data bus:** A group of connections used to route data. The LMM has three 16-bit data buses: *CB*, *DB*, and *EB*.

**data memory:** A memory region used for storing and manipulating data. Addresses 00h–1Fh of data memory contain CPU registers. Addresses 20h–5Fh of data memory contain peripheral registers.

**data page pointer (DP):** A 9-bit field in ST0 that specifies which of 512 128-word pages is currently selected for direct address generation. DP provides the nine MSBs of the data-memory address; the data memory address provides the lower seven bits. See also *direct memory address*.

**data ROM bit (DROM):** A bit in processor mode status register (PMST) that determines whether part of the on-chip ROM is mapped into program space.

**DB:** *D bus*. A bus that carries operands that are read from data memory.

**direct memory address (dma, DMA):** The seven LSBs of a direct-addressed instruction that are concatenated with the data page pointer (DP) to generate the entire data memory address. See also *data page pointer*.

**dma:** See *direct memory address*.

**DP:** See *data page pointer*.

**DROM:** See *data ROM bit*.

## E

**EAB address register (EAR):** A 16-bit register used to address the data memory operand represented by the immediate value in MVDK and MVMD instructions.

**EAR:** See *EAB address register*.

**EB:** *E bus*. A bus that carries data to be written to memory.

**exponent (EXP) encoder:** An application-specific hardware device that computes the exponent value of the accumulator.

## F

**fast return register (RTN):** A 16-bit register used to hold the return address for the fast return from interrupt (RETF[D]) instruction.

**fractional mode bit (FRCT):** A bit in status register ST1 that determines whether or not the multiplier output is left-shifted by one bit.

**FRCT:** See *fractional mode bit*.

## H

**HM:** See *hold mode bit*.

**hold mode bit (HM):** A bit in status register ST1 that determines whether the CPU enters the hold state in normal mode or concurrent mode.

## I

**IFR:** See *interrupt flag register*.

**IMR:** See *interrupt mask register*.

**instruction register (IR):** A 16-bit register used to hold a fetched instruction.

**interrupt:** A condition caused either by an event external to the CPU or by a previously executed instruction that forces the current program to be suspended and causes the processor to execute an interrupt service routine corresponding to the interrupt.

**interrupt flag register (IFR):** A 16-bit memory-mapped register used to identify and clear active interrupts.

**interrupt mask bit (INTM):** A bit in status register ST1 that globally masks or enables all interrupts.

**interrupt mask register (IMR):** A 16-bit memory-mapped register used to enable or disable external and internal interrupts. A 1 written to any IMR bit position enables the corresponding interrupt (when INTM = 0).

**interrupt service routine (ISR):** A module of code that is executed in response to a hardware or software interrupt.

**INTM:** See *interrupt mask bit*.

**IPTR:** *Interrupt vector pointer.* A 9-bit field in the processor mode status register (PMST) that points to the 128-word page where interrupt vectors reside.

**IR:** See *instruction register*.

**ISR:** See *interrupt service routine*.

## L

**latency:** The delay between when a condition occurs and when the device reacts to the condition. Also, in a pipeline, the delay between the execution of two instructions that is necessary to ensure that the values used by the second instruction are correct.

**LSB:** *Least significant bit.* The lowest order bit in a word.

## M

**memory-mapped register (MMR):** The '54x processor registers mapped into page 0 of the data memory space.

**microcomputer mode:** A mode in which the on-chip ROM is enabled and addressable.

**microprocessor mode:** A mode in which the on-chip ROM is disabled.

**micro stack:** A stack that provides temporary storage for the address of the next instruction to be fetched when the program address generation logic is used to generate sequential addresses in data space.

**MP/MC bit:** A bit in the processor mode status register (PMST) that indicates whether the processor is operating in microprocessor or microcomputer mode. See also *microcomputer mode*; *microprocessor mode*.

**MSB:** *Most significant bit.* The highest order bit in a word.



**O**

**OVA:** *Overflow flag A.* A bit in status register ST0 that indicates the overflow condition of accumulator A.

**OVB:** *Overflow flag B.* A bit status register ST0 that indicates the overflow condition of accumulator B.

**overflow:** A condition in which the result of an arithmetic operation exceeds the capacity of the register used to hold that result.

**overflow flag (OVA, OVB):** A flag that indicates whether or not an arithmetic operation has exceeded the capacity of the corresponding accumulator. See also *OVA* and *OVB*.

**overflow mode bit (OVM):** A bit in status register ST0 that specifies how the ALU handles an overflow after an operation.

**OVLY:** See *RAM overlay bit*.

**OVM:** See *Overflow mode bit*.

**P**

**PAB:** *Program address bus.* A 16-bit bus that provides the address for program memory reads and writes.

**PAGEN:** See *program address generation logic*.

**PAR:** See *program address register*.

**PB:** *Program bus.* A bus that carries the instruction code and immediate operands from program memory.

**PC:** See *program counter*.

**pipeline:** A method of executing instructions in an assembly-line fashion.

**pmad:** *Program-memory address.* A register that provides the address of a multiplier operand that is contained in program memory.

**PMST:** See *processor mode status register*.

**pop:** Action of removing a word from a stack.

**processor mode status register (PMST):** A 16-bit status register that controls the memory configuration of the device. See also *ST0*; *ST1*.

**program address generation logic (PAGEN):** Logic circuitry that generates the address for program memory reads and writes, and the address for data memory in instructions that require two data operands. This circuitry can generate one address per machine. See also *data address generation logic*.

**program address register (PAR):** A 16-bit register used to address the program-memory operands in FIRS, MACD, MACP, MVDP, MVPD, READA, and WRITA instructions.

**program controller:** Logic circuitry that decodes instructions, manages the pipeline, stores status of operations, and decodes conditional operations.

**program counter (PC):** A 16-bit register that indicates the location of the next instruction to be executed.

**program counter extension register (XPC):** A register that contains the upper 7 bits of the current program memory address.

**program data bus (PB):** A bus that carries the instruction code and immediate operands from program memory.

**program memory:** A memory region used for storing and executing programs.

**push:** Action of placing a word onto a stack.

## R

**RAM overlay bit (OVLY):** A bit in the processor mode status register PMST that determines whether or not on-chip dual-access RAM is mapped into the program/data space.

**RC:** See *repeat counter*.

**REA:** See *block-repeat end address*.

**register:** A group of bits used for temporarily holding data or for controlling or specifying the status of a device.

**repeat counter (RC):** A 16-bit register used to specify the number of times a single instruction is executed.

**reset:** A means of bringing the CPU to a known state by setting the registers and control bits to predetermined values and signaling execution to start at a specified address.

**RSA:** See *block-repeat start address*.

**RTN:** See *fast return register*.

**S**

**SARAM:** *Single-access RAM*. Memory that only can be read from or written during one clock cycle.

**shifter:** A hardware unit that shifts bits in a word to the left or to the right.

**sign-control logic:** Circuitry used to extend data bits (signed/unsigned) to match the input data format of the multiplier, ALU, and shifter.

**sign extension:** An operation that fills the high order bits of a number with the sign bit.

**sign-extension mode bit (SXM):** A bit in status register ST1 that enables sign extension in CPU operations.

**SINT:** See *software interrupt*.

**software interrupt:** An interrupt caused by the execution of an INTR instruction.

**SP:** See *stack pointer*.

**ST0:** *Status register 0*. A 16-bit register that contains '54x status and control bits. See also *PMST*; *ST1*.

**ST1:** *Status register 1*. A 16-bit register that contains '54x status and control bits. See also *PMST*; *ST0*.

**stack:** A block of memory used for storing return addresses for subroutines and interrupt service routines and for storing data.

**stack pointer (SP):** A register that always points to the last element pushed onto the stack.

**SXM:** See *sign-extension mode bit*.

**T**

**TC:** See *test/control flag bit*.

**temporary register (T):** A 16-bit register that holds one of the operands for add, load, multiply, store, and subtract instructions, the dynamic shift count for the add and subtract instructions, or the dynamic bit position for the bit test instructions.

**test/control flag bit (TC):** A bit in status register ST0 that is affected by test operations.

**transition register (TRN):** A 16-bit register that holds the transition decision for the path to new metrics to perform the Viterbi algorithm.

## W

**warm boot:** The process by which the processor transfers control to the entry address of a previously-loaded program.

## X

**XF:** *XF status flag.* A bit in status register ST1 that indicates the status of the XF pin.

**XPC:** See *program counter extension register*.

## Z

**ZA:** *Zero detect bit A.* A signal that indicates when accumulator A contains a 0.

**ZB:** *Zero detect bit B.* A signal that indicates when accumulator B contains a 0.

**zero detect:** See *ZA* and *ZB*.

**zero fill:** A method of filling the low- or high-order bits with zeros when loading a 16-bit number into a 32-bit field.

# Index

## A

ABDST instruction 4-2  
ABS instruction 4-3  
accumulator A D-1  
accumulator A high word (AH) D-1  
accumulator A low word (AL) D-1  
accumulator B D-1  
accumulator B guard bits (BG) D-2  
accumulator B high word (BH) D-2  
accumulator B low word (BL) D-2  
accumulator guard bits (AG) D-1  
accumulator shift mode (ASM) D-1  
accumulators D-1  
ADD instruction 4-4  
add instructions 2-2 to 2-3  
ADDC instruction 4-8  
ADDM instruction 4-9  
address D-1  
address visibility mode bit (AVIS) D-1  
addressing mode D-1  
ADDS instruction 4-10  
AND instruction 4-11  
description 4  
AND instructions 2-8  
ANDM instruction 4-13  
application-specific instructions 2-7  
AR0–AR7. *See* auxiliary registers  
ARAU. *See* auxiliary register arithmetic unit  
arithmetic logic unit (ALU) D-2  
arithmetic operation instructions 2-2  
add instructions 2-2 to 2-3  
application-specific instructions 2-7  
double (32-bit operand) instructions 2-6  
multiply instructions 2-4

multiply-accumulate instructions 2-4 to 2-5  
multiply-subtract instructions 2-4 to 2-5  
subtract instructions 2-3 to 2-4

ARP. *See* auxiliary register pointer  
ASM. *See* accumulator shift mode bits  
assembly language instructions 4-1  
auxiliary register arithmetic unit (ARAU) D-2  
auxiliary register file D-2  
auxiliary register pointer (ARP) D-2  
auxiliary registers (AR0–AR7) D-2  
AVIS. *See* address visibility mode bit

## B

B. *See* accumulator B  
B instruction 4-14  
BACC instruction 4-15  
BACCD instruction 4-15  
BANZ instruction 4-16  
BANZD instruction 4-16  
barrel shifter D-2  
BC instruction 4-18  
BCD instruction 4-18  
BD instruction 4-14  
description 5  
BIT instruction 4-21  
BITF instruction 4-22  
BITT instruction 4-23  
block-repeat active flag (BRAf) D-2  
block-repeat counter (BRC) D-2  
block-repeat end address register (REA) D-3  
block-repeat start address register (RSA) D-3  
boot D-3  
boot loader D-3  
branch instructions 2-10  
BRC. *See* block-repeat counter  
butterfly D-3

## C

C address bus (CAB) D-3  
 C bus (CB) D-3  
 C16 D-3  
 CALA instruction 4-25  
 CALAD instruction 4-25  
 CALL instruction 4-27  
 call instructions 2-11  
 CALLD instruction 4-27  
 carry bit (C) D-3  
 CC instruction 4-29  
 CCD instruction 4-29  
 CMPL instruction 4-32  
 CMPM instruction 4-33  
 CMPR instruction 4-34  
 CMPS instruction 4-35  
 CMPT. *See* compatibility mode bit  
 CNF. *See* configuration control bit  
 code D-3  
 cold boot D-3  
 compatibility mode bit (CMPT) D-3  
 compiler mode bit (CPL) D-3  
 conditional instructions  
   conditions A-2  
   grouping of conditions A-3  
 conditional store instructions 2-16  
 configuration control bit (CNF) D-4  
 contacting Texas Instruments vii  
 CPL. *See* compiler mode bit

## D

D address bus (DAB) D-4  
 D bus (DB) D-4  
 DAB address register (DAR) D-4  
 DADD instruction 4-37  
   description 3  
 DADST instruction 4-39  
 DAGEN. *See* data address generation logic  
 DAR. *See* DAB address register  
 DARAM 3-1  
 data address bus D-4  
 data address generation logic (DAGEN) D-4

data bus D-4  
 data memory D-4  
 data page pointer (DP) D-4  
 data ROM bit (DROM) 4-4  
 DELAY instruction 4-41  
 direct memory address D-4  
 DLD instruction 4-42  
 double (32-bit operand) instructions 2-6  
 DP. *See* data page pointer  
 DROM 3-1  
   *See also* data ROM bit  
 DRSUB instruction 4-43  
 DSADT instruction 4-45  
 DST instruction 4-47  
 DSUB instruction 4-48  
 DSUBT instruction 4-50  
 dual-access RAM (DARAM) D-4

## E

E bus (EB) D-5  
 EAB address register (EAR) D-5  
 EAR. *See* EAB address register  
 EXP encoder D-5  
 EXP instruction 4-52  
 exponent encoder D-5

## F

fast return register (RTN) D-5  
 FB instruction 4-53  
 FBACC instruction 4-54  
 FBACCD instruction 4-54  
 FBD instruction 4-53  
 FCALA instruction 4-55  
 FCALAD instruction 4-55  
 FCALL instruction 4-57  
 FCALLD instruction 4-57  
 finite impulse response (FIRS) filter instruction 4-59  
 FIRS instruction 4-59  
 fractional mode bit (FRCT) D-5  
 FRAME instruction 4-60  
 FRCT. *See* fractional mode bit  
 FRET instruction 4-61  
 FRET D instruction 4-61

FRETE instruction 4-62  
 FRETED instruction 4-62

## H

HM. *See* hold mode bit  
 hold mode bit (HM) D-5

## I

IDLE instruction 4-63  
 IFR. *See* interrupt flag register  
 IMR. *See* interrupt mask register  
 instruction cycles, assumptions 3-2  
 instruction register (IR) D-5  
 instruction set  
   abbreviations 1-2  
   classes 3-3 to 3-72  
   cycle tables 3-3 to 3-72  
   example description 1-9  
   notations 1-7  
   opcode abbreviations 1-5  
   opcode symbols 1-5  
   operators 1-8  
   symbols 1-2  
 instruction set summary  
   add instructions 2-2 to 2-3  
   AND instructions 2-8  
   application-specific instructions 2-7  
   branch instructions 2-10  
   call instructions 2-11  
   conditional store instructions 2-16  
   double (32-bit operand) instructions 2-6  
   interrupt instructions 2-11  
   load instructions 2-14 to 2-15  
   miscellaneous load-type and store-type instructions 2-18  
   miscellaneous program control instructions 2-13  
   multiply instructions 2-4  
   multiply-accumulate instructions 2-4 to 2-5  
   multiply-subtract instructions 2-4 to 2-5  
   OR instructions 2-8  
   parallel load and multiply instructions 2-16  
   parallel load and store instructions 2-16  
   parallel store and add/subtract instructions 2-17  
   parallel store and multiply instructions 2-17  
   repeat instructions 2-12

return instructions 2-12  
 shift instructions 2-9  
 stack-manipulating instructions 2-13  
 store instructions 2-15  
 subtract instructions 2-3 to 2-4  
 test instructions 2-9  
 XOR instructions 2-9

### interrupt

  definition D-5  
   locations B-1  
   priorities B-1  
 interrupt flag register (IFR)  
   definition D-6  
   figure C-3  
 interrupt instructions 2-11  
 interrupt mask bit (INTM) D-6  
 interrupt mask register (IMR)  
   definition D-6  
   figure C-4  
 interrupt service routine (ISR) D-6  
 interrupt vector pointer (IPTR) D-6  
 INTM. *See* interrupt mask bit  
 INTR instruction 4-65  
 IR. *See* instruction register  
 ISR. *See* interrupt service routine

## L

latency D-6  
 LD instruction 4-66, 4-70  
 LD||MAC instruction 4-74  
 LD||MACR instruction 4-74  
 LD||MAS instruction 4-76  
 LD||MASR instruction 4-76  
 LDM instruction 4-73  
 LDR instruction 4-78  
 LDU instruction 4-79  
 least significant bit (LSB) D-6  
 LMS instruction 4-80  
 load and store operation instructions 2-14  
   conditional store instructions 2-16  
   load instructions 2-14 to 2-15  
   miscellaneous instructions 2-18  
   parallel load and multiply instructions 2-16  
   parallel load and store instructions 2-16  
   parallel store and add/subtract instructions 2-17  
   parallel store and multiply instructions 2-17

- store instructions 2-15
- load instructions 2-14 to 2-15
- logical operation instructions 2-8
  - AND instructions 2-8
  - OR instructions 2-8
  - shift instructions 2-9
  - test instructions 2-9
  - XOR instructions 2-9
- LTD instruction 4-81

## M

- MAC instruction 4-82
- MACA instruction 4-85
- MACAR instruction 4-85
- MACD instruction 4-87
- MACP instruction 4-89
- MACR instruction 4-82
- MACSU instruction 4-91
- MAR instruction 4-92
- MAS instruction 4-94
- MASA instruction 4-97
- MASAR instruction 4-97
- MASR instruction 4-94
- MAX instruction 4-99
- memory-mapped register (MMR) D-6
- micro stack D-6
- microcomputer mode D-6
- microprocessor mode D-6
- MIN instruction 4-100
- miscellaneous load-type and store-type instructions 2-18
- miscellaneous program control instructions 2-13
- MMR 3-1
- most significant bit (MSB) D-6
- MP/MC bit D-6
- MPY instruction 4-101
- MPYA instruction 4-104
- MPYR instruction 4-101
- MPYU instruction 4-106
- multi-cycle instructions, transformed to single-cycle 2-19
- multiply instructions 2-4
- multiply-accumulate instructions 2-4 to 2-5
- multiply-subtract instructions 2-4 to 2-5

- MVDD instruction 4-107
- MVDK instruction 4-108
- MVDM instruction 4-110
- MVDP instruction 4-111
- MVKD instruction 4-113
- MVMD instruction 4-115
- MVMM instruction 4-116
- MVPD instruction 4-117

## N

- NEG instruction 4-119
- nonrepeatable instructions 2-20
- NOP instruction 4-121
- NORM instruction 4-122

## O

- OR instruction 4-123
- OR instructions 2-8
- ORM instruction 4-125
- OVA. *See* overflow flag A
- OVB. *See* overflow flag B
- overflow D-7
- overflow flag D-7
- overflow flag A (OVA) D-7
- overflow flag B (OVB) D-7
- overflow mode bit (OVM) D-7
- OVLY. *See* RAM overlay bit
- OVM. *See* overflow mode bit

## P

- PAGEN. *See* program address generation logic
- PAR. *See* program address register
- parallel load and multiply instructions 2-16
- parallel load and store instructions 2-16
- parallel store and add/subtract instructions 2-17
- parallel store and multiply instructions 2-17
- PC. *See* program counter
- pipeline D-7
- pmad D-7
- PMST. *See* processor mode status register
- POLY instruction 4-126
- pop D-7



POPD instruction 4-127  
 POPM instruction 4-128  
 PORTR instruction 4-129  
 PORTW instruction 4-130  
 processor mode status register (PMST)  
   definition D-7  
   figure C-5  
 program address bus (PAB) D-7  
 program address generation logic (PAGEN) D-8  
 program address register (PAR) D-8  
 program bus (PB) D-7  
 program control operation instructions 2-10  
   branch instructions 2-10  
   call instructions 2-11  
   interrupt instructions 2-11  
   miscellaneous instructions 2-13  
   repeat instructions 2-12  
   return instructions 2-12  
   stack-manipulating instructions 2-13  
 program controller D-8  
 program counter (PC) D-8  
 program counter extension (XPC) D-8  
 program data bus (PB) D-8  
 program memory D-8  
 program memory address (pmad) D-7  
 PROM 3-1  
 PSHD instruction 4-131  
 PSHM instruction 4-132  
 push D-8

## R

RAM overlay bit (OVLY) D-8  
 RC. *See* repeat counter  
 RC instruction 4-133  
 RCD instruction 4-133  
 REA. *See* block-repeat end address  
 READA instruction 4-136  
 register D-8  
 repeat counter (RC) D-8  
 repeat instructions 2-12  
 repeat operation 2-19  
   handling multicycle instructions 2-19  
   nonrepeatable instructions 2-20  
 reset D-8

RESET instruction 4-138  
 RET instruction 4-139  
 RETD instruction 4-139  
 RETE instruction 4-140  
 RETED instruction 4-140  
 RETF instruction 4-141  
 RETFD instruction 4-141  
 return instructions 2-12  
 RND instruction 4-142  
 ROL instruction 4-143  
 ROLTC instruction 4-144  
 ROM 3-1  
 ROR instruction 4-145  
 RPT instruction 4-146  
 RPTB instruction 4-148  
 RPTBD instruction 4-148  
 RPTZ instruction 4-150  
 RSA. *See* block-repeat start address  
 RSBX instruction 4-151  
 RTN. *See* fast return register

## S

SACCD instruction 4-152  
 SARAM 3-1  
 SAT instruction 4-154  
 SFTA instruction 4-155  
 SFTC instruction 4-157  
 SFTL instruction 4-158  
 shift instructions 2-9  
 shifter D-9  
 sign control logic D-9  
 sign extension D-9  
 sign-extension mode bit (SXM) D-9  
 single-access RAM (SARAM) D-9  
 SINT. *See* software interrupt  
 software interrupt D-9  
 SP. *See* stack pointer  
 SQDST instruction 4-160  
 SQUR instruction 4-161  
 SQURA instruction 4-163  
 SQURS instruction 4-164  
 SRCCD instruction 4-165  
 SSBX instruction 4-166

## Index

---

ST instruction 4-167  
ST||ADD instruction 4-177  
ST||LD instruction 4-178  
ST||MAC instruction 4-180  
ST||MACR instruction 4-180  
ST||MAS instruction 4-182  
ST||MASR instruction 4-182  
ST||MPY instruction 4-184  
ST||SUB instruction 4-185  
ST0. *See* status register 0  
ST1. *See* status register 1  
stack D-9  
stack pointer (SP) D-9  
stack pointer register 10  
stack-manipulating instructions 2-13  
status register 0 (ST0)  
    definition. *See* PMST, ST1  
    figure C-5  
status register 1 (ST1)  
    definition. *See* PMST, ST0  
    figure C-5  
STH instruction 4-169  
STL instruction 4-172  
STLM instruction 4-175  
STM instruction 4-176  
store instructions 2-15  
STRCD instruction 4-186  
SUB instruction 4-187  
SUBB instruction 4-191  
SUBC instruction 4-192  
SUBS instruction 4-194  
subtract instructions 2-3 to 2-4

SXM. *See* sign-extension mode bit

## T

TC. *See* test/control flag bit  
temporary register (T) D-9  
test instructions 2-9  
test/control flag bit (TC) D-10  
transition register (TRN) D-10  
TRAP instruction 4-195

## W

warm boot D-10  
WRITA instruction 4-196

## X

XC instruction 4-198  
XF status flag (XF) D-10  
XOR instruction 4-201  
XOR instructions 2-9  
XORM instruction 4-203  
XPC. *See* program counter extension register

## Z

zero detect. *See* zero detect bit A; zero detect bit B  
zero detect bit A (ZA) D-10  
zero detect bit B (ZB) D-10  
zero fill D-10

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☒ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**